UNIVERSITY OF OSLO
Department of Informatics

Design of a CMDB with integrated knowledge management based on Topic Maps

Master thesis

Fitim Haziri

Network and System Administration

Oslo University College

May 21, 2009

Design of a CMDB with integrated knowledge management based on Topic Maps

Fitim Haziri

Network and System Administration
Oslo University College

May 21, 2009

Abstract

Configuration management databases have gained popularity in enterprises due to their role in providing efficient IT Resource and Service Management. Enterprises are becoming more competitive through increasing of resource utilization to support their business services. Existing configuration management database implantations are known to have serious problems, introducing security and maintenance issues. They use a centralized approach implemented via a complex logical database model. This complexity reduces the possibility for enterprises to achieve competitive advantage. Apart from this, implementing such a complex model requires time.

There is room for a new logical database model. Cfengine's approach to logical database is not as a traditional inventory, but rather as a knowledge-base semantic web of information that connects various aspects of configuration management. The thesis considers designing of a logical database model, and its topic map model for Cfengine 3, which is a machine-learning approach. The developed model is characterized of being easily manageable, easy to implement, extensible, and optimized for updating process.

Acknowledgements

I am grateful for the teaching, encouragement, guidance, and support of my supervisor Professor Mark Burgess. I want to thank Steve Pepper for topic maps ontology overview, and Professor Alva L. Couch for his comments on my thesis work. Thanks to Associate Professor Harek Haugerud and Kyrre Begnum for beneficial discussions.

A special thanks to my family for support and encouragement.

The project idea of designing a Semantic CMDB originates from Professor Mark Burgess.

Contents

List of Figures

Chapter 1

Introduction

Most of existing configuration management database models for system management are derived from the Common Information Model (CIM). CIM is an object-oriented management model that provides a common way of representing information about IT components. CIM defines resources as object classes which can be further specialized by means of inheritance. CIM contains over 1000 predefined base classes. CIM is indeed a complex model. Understanding the relationships between classes and adopting CIM to the requirements of an organization invokes time-consuming efforts. Furthermore, much of the information stored in standard classes deals with low level details of resources, and does not even contain the level of abstraction required to represent IT Service and Management scope [1].

In object-oriented modeling one is forced to pursue a strict procedure, building a tree structured model. Behaviors are scattered across this hierarchy. The object-oriented model (as expressed in languages such as Java and C++) is rigid. When we add a new element (even though it is just an updated version of an existing one), we have to update the model as well. Updating of the hierarchy model with new constructs, at some point it will become unmanageable. An error in modeling or an incompatibility in enriching the model with new elements requires redesigning the model.

Different vendors have pursued various approaches in adopting CIM for their logical database solutions. HP implements only a subset of CIM, and its core model called Universal CMDB supports more than 200 classes. Meanwhile, IBM widely adopts CIM as its base model. IBM's version is referred to as the Common Data Model (CDM). It provides about 1000 classes, subclasses, and relationships. Meanwhile, our approach is to design a logical database model, that incarnates certain distinctive characteristics compared to existing CMDB models. Above all it can be characterized as being simple, easily manageable, independent, extensible, and easy to implement.

These models seem far too complex to maintain and computationally expensive to use. Part of the trouble seems to be an adherence to object-oriented

dogma. So we wish to study alternatives that might avoid the pitfall of object-oriented modeling.

1.1 Motivation and research questions

Datacenters are dynamic environments, which means that the logical database must be updated often, to reflect the current state of the network environment. Taking this fact into account, our motivation was to design a logical database model that is optimized for updating.

The goal of the thesis was ultimately

- *Designing a logical database model that is much simpler than existing logical database models and simultaneously provides numerous capabilities*

To approach this goal we had to answer a number of questions

- *Could a semantic approach to knowledge address the complexity issues in traditional modeling ?*
- *Is there a simple relationship between topic maps and entity relation model databases ?*
- *Can we find an optimal model that will scale to tens of thousands of machines, common in today's datacenters ?*

1.2 Overview of the Thesis

- In chapter 2 we describe existing CMDB commercial solutions. CMDB vendors restrict materials on their core solution approach for the public. Despite this fact, we believe that we managed to touch the main elements of their approaches.
- Next in chapter 3, we compare features of topic maps and entity relation databases.
- In chapter 4 we describe the process of development of our CMDB model.
- In chapter 5 we describe the principles of our ultimate CMDB model. Designed the topic map ontology of the CMDB, and present the results.
- In chapter 6 we present a test case and we evaluate it. Then, we continue discussing our CMDB drafts. We finish discussion by comparing our ultimate CMDB model with star data warehouse model.
- In chapter 7 we conclude our research work and suggest future work.

Chapter 2

Background and literature

The Common Information Model is built upon an object-oriented (OO) model. CIM is designed to represent and organize the generic management information in an enterprise environment. It can be described using Unified Modeling Language (UML) diagrams. UML is an open standard language for specifying, visualizing, designing and documenting models [8]. CIM consists of a *schema* and a *specification*. The *schema* [5] is the actual model, which collects defined object classes representing enterprise's physical devices, software etc. While the *specification* defines the details for integration with the other management models [6]. The CIM model consists of over 1000 classes and associations defining different enterprise's components [5]. It is continuously under development, which means that the number of provided classes increases over time.

A CIM schema consists of three conceptual layers. Th *Core* model defines concepts that are applicable to all system managements. Based on selection of classes vendors made here, they determine how extended the *Common* schema would be. The *Common* schema provides models that describe particular management areas in a detail manner. They are applicable regardless of technology or implementation. The *Common* model includes, such as application, devices, users etc. They implement very detail scope, e.g. common device model among other addresses low level concepts such as fans, sensors, etc
The *Extension* schema is designed to handle technology-specific aspects of management information. CIM *Extension* models are specific to environments, such as operating systems. In addition, vendors can add their own classes to support their hardware and software products as a part of the CIM model.

CIM is a hierarchical, object-oriented information management model. It unifies and extends management standards, such as SNMP, CMIP, DMI, etc using object-oriented design. CIM is independent of technology, implementation, and does not require any persistent information repository format. It means that CIM is applicable on all kinds of applications.

This model has some disadvantages. The CIM model is very generic information model. It covers a wide scope of objects, which makes stored information unmanageable. Attributes declared in classes are atomic, this makes classes complex. Object-oriented property inheritance can make subclasses

difficult to understand. Subclasses often inherit unnecessary properties and behaviors from their superclasses that they do not need for doing their essential tasks. Not all types of information are suitable to be organized based on the CIM (Object-Oriented) model. Object-oriented model is hierarchical, meaning that it defines parent-child relationships. In the CIM model, as displayed in Figure 2.1, a relationship is modeled as a class that contains two or more references. Relationships modeled in CIM handle only class-level relationships [2].

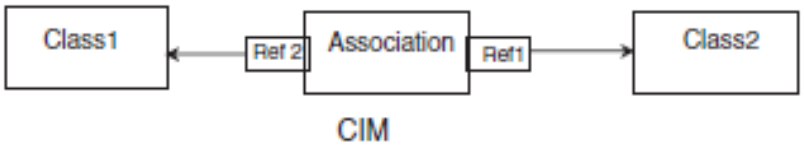

Figure 2.1: Class-level relationships modeled in CIM

Standard	Complexity of Concepts	Power of Concepts	Complexity of Usage
SNMP	Low	Low	High
ISO OSI Management Standards	High	High	High
DMI	Low	Low	Medium
CMI	Medium	High	Medium

Figure 2.2: Comparsion of CIM with other information models [3]

A comparison of CIM with other popular information models (DMI, ISO, SNMP, and OSI management standards), presented in [3], concluded that CIM is superior, but nevertheless it is characterized as a complex information model. This complexity is introduced due to the fact that CIM model incorporates an object-oriented model.
Large CMDB vendors, such as BMC, IBM, and HP adopted the CIM core model within their CMDB models.

Defination 1 *(Configuration management database (CMDB)). A CMDB reflects the current state of a datacenter's assets. CMDB is used for the purpose of increasing support for business services through defining, monitoring, and managing resource utilization.*

Since a CMDB is a logical representation of a datacenter's assets, we refer to it also as a logical database.

The fundamental feature of CMDB is that it has to be populated with data dynamically without interference of human factor. The autonomous population distinguishes a CMDB from the traditional databases. The autonomy notion implies high efficiency in collecting data and a low level of inconsistent data. Not all of the data can be captured and recorded into the logical database dynamically. E.g. population of the CMDB with data on people or physical location takes place manually by authorized people. This means that one must have at least a copy of manually-entered information physically stored somewhere.

2.1 HP CMDB Architecture

By choosing an object-oriented model, users are forced to confront a rigid model for CMDB every time they want to add new capabilities. This is both

a barrier to usage and a source of later trouble. Since object-oriented model uses a strict hierarchy, an error of modeling is very difficult to correct, because sub-class relationships are rigid, and an error can cause a cascade of errors throughout the hierarchy. Maintenance is thus potentially exponentially expensive.

HP announced its CMDB solution in 2004, and since then HP has advanced and modified its CMDB model. The HP Universal CMDB class model is built upon the CIM (object-oriented) model. HP's core class model supports more than 200 common IT constructs and relationships between them. Each class has a long list of attributes. e.g Unix class has more that 60 attributes. An instance of a class stores too detail information, in addition information is unclassified.

HP claims that its CMDB model is customizable, allowing enterprises to adopt, or evolve their CMDB, according to their needs. HP provides a graphical user interface (GUI) program to modify the CMDB model interactively. The philosophy that is using object-oriented properties, such as inheritance, and polymorphism, the HP CMDB reuses existing code to create new classes, or delete or add new attributes. Nevertheless, users need advanced programming skills in order to modify or extend the HP CMDB schema.

Meanwhile, the flexibility of CMDB model is limited by discovery model. It is a pure agentless model, which means that its discovering capabilities are limited, therefore we have a reflection on CMDB model.

2.1.1 Discovery process

Apart from providing Universal CMDB (UCMDB), HP provides also Discovery and Dependency Mapping tool (DDM). Their aim is to discover devices and applications' components inside IT infrastructure, populating CMDB with current configuration information and map relationships of IT components. The DDM tool utilizes patterns to discover Configuration Items (CI) on the IT environment. Patterns are kept in a library, they can be altered, deactivated, deleted, and new patterns can be added to meet the enterprises' needs.
The Discovery and Dependency Mapping tool support two discovery mechanisms:

- *Passive discovery*
- *Active discovery*

HP's discovery process is referred to as a *spiral* process, due to the fact that the outcome of one discovery pattern can initiate another discovery pattern. As an example, the pattern that captures TCP connections from a server machine will create Configuration Items (CI) for those machines to which the target server is connected. If the system administrator has activated a discovery pattern for gathering detailed machine information, the creation of the CIs would invoke the detailed host discovery against those new discovered machines. The spiral discovery has two stages. The first stage allows the DDM to learn about related CIs. The example above with TCP connections introduces

the first type of discovery. This process is known as *widening* the spiral. The second stage gathers more detail information about the new discovered machines. As an example a list of running services and applications. This process is known as *deepening* the spiral.

In our opinion, the *spiral* model introduces drawbacks in the discovery process. It discovers machines in series, but the series imply dependency. Thus, if the server that the DDM has in its list is unavailable, DDM cannot discover client machines. So, the absence or even temporary unavailability of one component will influence the list of the rest of the components. This propagates the failure in scope and depth.

Defination 2 *(Configuration Item). The notion of configuration item (CI) represents the current configuration state of an entity in the IT environment. Kinds of entities include: Physical (machine, router etc), Logical (an instance of a software program),Conceptual (a business service).*

The agentless model is a centralized approach. Its name might sound appealing (no need to install agents on machines), but it depends on the availability of agents from other vendors. The agentless model has to be integrated with different agents designed with various specifics, making very hard to deploy and maintain. In addition, it does not scale on large networks, and support the *timescale* as feature. The agent based model is a decentralized approach, making it scalable in the large networks. An agent installed on the machines provides real time data. The agent based model is more secure than agentless and has richer capabilities.

2.1.2 Passive discovery

Passive monitoring (as its name implies) operates simply just by observing network traffic. By means of IP accounting, provided by Cisco NetFlow technology, HP Discovery and Dependency Mapping (DDM) gains access to network flow through a network device such as a router or switch. The data that it captures is quite minimal, including only IP addresses and standard port numbers. In case IT services are running on non-standard ports, passive or active discovery will not discover services unless these changes are entered manually into the discovering tool. Passive monitoring has huge limitations. It cannot obtain detailed configuration information, and is limited by the scope and depth of traffic passing through the observed network devices. It is not able to observe traffic flow on local machines. As an example, if an Apache web server is the only client of a PostgreSQL database on the same machine, the traffic flow between them never passes over network devices, so passive discovery would never learn of the existence of the PostgreSQL database or Apache web server. Encrypted traffic cannot be decoded by passive monitoring, it is useless as a discovery aid. IP spoofing can fool passive mapping to

conclude incorrectly that there are relationships between hosts. This breaks the entire agentless model, since an active discovery probe uses IP addresses and port numbers gathered by passive monitoring to query machines.

2.1.3 Active discovery

Active discovery obtains configuration information by running queries against standard interfaces on service endpoint hosts. It can discover only information that is accessible over the network -unless local agents are used that continue to work when offline.

Before active discovery gathers data from the endpoint hosts, HP's discovering component must be configured with credentials. The discovering component requires credentials for every host and application for which we want to gather information.

HP's active discovery requires also more configuration to be done on network devices and end hosts. As example is that in order for DDM to discover configuration information of a switch, it uses the SNMP protocol and "READ" instruction for the network device. In addition, the switch must have DDM in its Access Control List (ACL), otherwise retrieving of configuration information will fail. Meanwhile, the procedure on endpoint machines is even more complex. In addition to logging into machines and applications, personal firewall and applications have to be configured to allow (agentless) DDM to retrieve applications' configuration and other OS specifics.

2.1.4 Assessment of agentless model

The main drawback of agentless models is that they must gain root access to the hosts in order to collect the most important information. This leads to difficulties in how we manage and maintain secure root, application and database passwords for every machine. As an example, if a database requires a password for logging in, the agentless model can not discover the database's components, unless it has the password in its internal database. This is in conflict with enterprises' security policies. System administrators do not want to distribute database passwords on other network devices, since databases contain highly classified information such as credit card numbers. Distributing passwords on different network devices makes passwords more vulnerable, since malicious people have different places to try. It has been widely proven that retaining and managing sensitive information is a daunting task, especially when it is distributed in many places.

The active agentless model has impact on network load, and therefore it cannot provide a continuous and real time discovery of IT components. Agentless models can discover only services and applications that are running. They are not able to discover services or applications that are installed and not running due to configuration errors, or other issues. The active agentless models are dependent on running some services on target hosts. The agentless model cannot control these services, if they are not running or they block remote access, the active agentless model cannot gather information.

A fundamental drawback of the active agentless model is that it is difficult, and costly to integrate with applications and services. To carry out its tasks, the active agentless model always is dependent on input data, such as passwords, IP addresses and port numbers. The agentless model does not have *timescale* as feature. Consequently, it queries machines for data that already exists in the CMDB, or the configuration state of machines changes while old data resides in the CMDB.

2.1.5 HP CMDB Visualization and mapping

One of the key features of a CMDB implementation is the visualization and mapping model that is applied. A traditional CMDB contains a lot of detailed information, some of which is quite frequently updated. In order to get benefits of this data, we need a powerful visualization and mapping model, which will give us the means to extract any kind of information from CMDB. The visualization of data starts after the CMDB is populated with data.

HP's visualization and mapping approach is by dragging and dropping icons that represent the CI types present in the HP Universal CMDB. This drag-and-drop approach is used to create templates that yield the desired query results. We use temples to query the HP Universal CMDB, and CMDB replies by returning all the results that match the criteria described on the template. The results are displayed graphically as a view, or output to a report. As an example, a system administrator drags and drops CI icons to create a query to find all servers running Oracle database. Figure 2.3 shows an example query and its results for hosts running Oracle database.

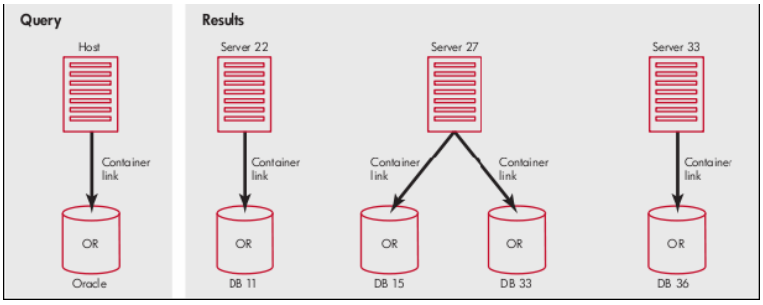

Figure 2.3: Extraction of data from UCMDB [4]

As we can see from the Figure 2.3, HP's CMDB model provides only what it has been queried for. It just displays servers running Oracle database, not any other information that might be related with server or database application. Creation of simple queries is trivial, however, creating more complex queries requires detailed knowledge of applications and their dependencies. In addition, the HP's solution lacks a full package for describing all applications.

The conclusion that we draw here is that the HP's query mechanism is equivalent to relational databases. The result contains only the objects that match the criteria, it does not provide other objects that have physical or logical associations with the matched objects. Therefore, HP's query model is not intelligent to deliver associated data that we might be interested in.

2.1.6 Federation

IT environments evolve continuously, in relation with enterprises' business services. Diversity of business services sometimes requires one to implement different types of technologies and systems. This diversity of technologies leads to storing data on different location and technology platforms. Linking together these disparate data source is indispensable, since it facilitates enterprises to fully leverage existing investments. Federation is a functionality of CMDB that tie together data residing in different systems without the need to copy or replicate all data from one system to another. Federation helps customers automate their IT environments and have greater view and control over multi-vendor IT environments. HP provides federation and reconciliation capabilities to its products and third party management solutions. Federation to various data sources can be enabled through the use of the HP Universal CMDB web-services-based software development kit (SDK). But, the drawback of this approach is that it is required to write code for federating data.

2.2 Cfengine

Cfengine is an agent-based configuration management system developed at Oslo University College. Cfengine provides a framework for researching various topics in the field of configuration management and system monitoring.

Cfengine's characteristics, quoted from [19]:

- Centralized policy-based specification, using an operating system independent language.
- Distributed agent-based action; each host agent is responsible for its own maintenance.
- Convergent semantics encourage every transaction to bring the system closer to an ideal averagestate, like a ball rolling into a potential well.
- Once the system has converged, action by the agent desists.

The components of cfengine:

Cfengine is a complex program and it consists of several program tools. These programs (cf-servd, cf-agent, cf-execd, cf-run, cf-key, cf-envd) are installed on each host under a local directory /var/cfengine/sbin/. The path is local to each host, and it makes sure that they are available at any time.

cf-serverd: The main task of cf-servd is to make the policy configuration files accessible to client hosts. It is a center of security for cfengine and should be locked down as tightly as possible. It enables also the cfengine agent to be started remotely.

cf-agent: The core of the cfengine program is cf-agent which is a autonomous configuration agent. Its function is to reconfigure the host to the desired state based on the policy of the configuration file cfagent.conf. In a large network this file can become very complex and difficult to administrate, therefore the system administrator breaks it up into multiple files. cf-agent can be started through a cf-execd wrapper, a cron timer or manually. From a cron timer it is common to start cf-agent with option F.

cf-execd: This is a daemon which can be used for controlling cfengine execution. cf-execd's default configuration is to execute cf-agent every hour. It is equivalent to the cron program from Unix/Linux.

cf-monitord: This is a client-side environment daemon which gathers data about the host and adds the host to certain classes.

cf-know: It is a knowledge management component in which topic map ontologies can by learned and stored.

cf-promises: It is a syntax and error checking module that validates the cfengine configuration by describing its impact upon configuration.

cf-runagent: This program simulates the push model by executing cf-agent remotely. To run cf-agent on target host, the cf-execd daemon is required to be running.

cf-key: Generates public/private keys pairs on the local host. They are used to establish mutual authentication between server and client. Local hosts public-private keys and other systems public key are retained under ppkey/ directory.

2.3 IBM CMDB Architecture

IBM widely adopts CIM as its CMDB model. Currently, IBM's version support around 1000 classes, subclasses, and relationships [5]. These object classes and relationships are implemented using Java persistent objects.
A disadvantage of IBM's solution is that one needs to be a good java programmer in order to manipulate objects and relationships.

IBM provides an API named Model Query Language (MQL) for quiring CMDB. It is expressed in an SQL-like language. The MQL filter introduces two issues:

- Not all configuration queries can be conducted using this filter
- Delays occur, because it is required that CCMDB translate the expressed filers into Structured Query Language (SQL) statements.

End-users need to understand the CCMDB schema in order to construct queries. They must know where exactly the data is stored (in which table), join tables etc. Consequently, they end up constructing complex queries. The follwing query is qouted from [13], it searches for IBM machines that run Windows XP as operating system, and have installed Norton Antivirus 1.6:

```
SELECT *
FROM ComputerSystem
WHERE ComputerSystem.physicalPackage.
manufacturer == 'IBM'
AND OSRunning.OSName == 'Windows XP'
AND exists(OSRunning.installedSoftware.
productName == 'Norton Antivirus 1.6')
```

Tivoli Application and Dependency Manager (TADDM) is a software application that discovers applications, and maps their relationships and dependencies [6]. The discovery process is again an agentless approach, consequently it has same drawbacks as HP's agentless approach. Meanwhile, there is a difference in the way IBM, and HP build their IP databases. IBM's discovery approach is not vulnerable to IP spoofing (it does not implement HP's method of building IP database via monitoring network traffic), since it requires one to enter the IP range manually inside the database.

2.4 BMC Atrium CMDB

BMC designed its CMDB by adopting CIM and implementing ITIL recommendations for building CMDB. ITIL define the configuration management database (CMDB) as a single configuration repository, mirroring the actual state of IT components.

Before changes are registered in the BMC CMDB Atrium, a user must verify manually each of the changes, for the purpose of discovering unauthorized changes. After the change is accordance with policy, then the user sends it into the Atrium CMDB. e.g. if we update 100 windows machines with the latest patches, the user has to process all this information. Dunbar's research finding exhibits that humans can cope with a limited amount of information [7]. This means that the users might take wrong decision while analyzing the stream of information, and consequently Atrium CMDB will not be accurate.

2.4.1 Web services

BMC Discover for Business Processes discovers web services with their properties, such as the application server that delivers web services (Apache, Jboss), port number and hosting machine. It also shows dependencies and hosting relationships. But the drawback is that end users are supposed to input the URL of web services to the discovery process.

2.4.2 Summary

To summarize the drawbacks of active agentless model, we list the following issues: requirements for storing of sensitive information (passwords) on distributed network devices, inability to integrate with applications and services, dependence on services running on target hosts, limited autonomous functionality, does not run continuously and as a consequence, does not provide the configuration state of hosts in real time, can discover only services and applications that are running, and does not support the *timescale* feature.

Chapter 3

Comparison of topic maps and entity relation databases

This chapter gives a short comparison between Topic Maps and Entity Relation models. The aim of conducting this evaluation was to highlight the strengths and weaknesses of these information models, so that we adopt only their strengths in our CMDB model, and avoid pitfalls.

3.1 What is ontology ?

Our relationships are based on types, which in turn are naming conventions for sets of entities. How we choose to name groups has an impact on the resulting design. Ontology is the study of naming of concepts. It has its origin in philosophy. Ontology discusses the categories and relationships of various modes or being that exist. The ontology concept is broad, which has found usage in many scientific fields, such as in Artificial Intelligence, Computer Science, Software Engineering, Knowledge Engineering etc. Therefore, there are numerous definitions of ontologies. According to Gruber the ontology term in context of computer science means [8]:

"Ontology is a description of the concepts and relationships that can exist for an agent or a community of agents."

Despite the fact that different scientists of different fields formulate the meaning of ontology in different contexts, nevertheless there are elements of similarities. e.g. computer science and philosophy have in common categories, entities, relationships between entities, events etc. Generally, the differences between representing the entites is the matter of nature of the fields.

While John Strassner in [9] defines ontologies for network and system administration as:

"An ontology is a formal, explicit specification of a shared, machine-readable vocabulary and meanings, in the form of various entities and relationships between them, to describe knowledge about the contents of one or more related subject domains

throughout the life cycle of its existence. These entities and relationships are used to represent knowledge in the set of related subject domains. Formal refers to the fact that the ontology should be representable in a formal grammar. Explicit means that the entities and relationships used, and the constraints on their use, are precisely and unambiguously defined in a declarative language suitable for knowledge representation. Shared means that all users of an ontology will represent a concept using the same or equivalent set of entities and relationships. Subject domain refers to the content of the universe of discourse being represented by the ontology."

Defination 3 *(Ontology). Ontology is a way of describing the objects that exist.*

3.2 The topic map model

Topic maps were originally designed for representing knowledge structures as traditional book-indices. Possibilities for topic maps include more than being a digital form of book-indices. Topic maps define the associations between concepts in indices. The topic map model is an ISO standard for the representation and interchange of knowledge [10]. Topic maps have been called as the "GPS (Global Positioning System) of the information universe", as they are designed to increase navigation in complex information pools. Topic maps fill the gap between the abstract world and real world, linking together high level and low level views of information.

Topic Maps collect all various concepts of information and bind them together in order to describe the relationships among them. Topic maps can tie together different types of information such as electronic documents, images, diagrams and databases.

The topic map model is constructed of the three layers: types, topics and occurrences. Each of them labels a different level of information granularity [11].

3.2.1 Topics

The topic is the base concept in topic map model. A topic is a representation of any subject, one wants to discuss regardless of whether it is physical or imaginary. e.g. a given topic might represent a person, a machine, or an idea. It is an aggregation of topic properties. In the real world, objects might have multiple names, e.g. a machine is labeled also as host or computer. The topic map model handles it, allowing us to capture objects through their synonyms. This feature is not supported by entity relation and object-oriented models. A topic in a topic map is the smallest unit of comprehensive knowledge in the map. In the topic map model the relationship between topics and subjects is of type one-to-one. It ensures that all knowledge about a particular subject can be accessed via a single topic.

3.2.2 Topic types

Topics can be grouped into disjoint topic-types, so that we can collate things according to their intentions. The topic-type is itself a topic, wich serves as a container. Two different things can have same concept name, but completely unrelated intentions. Types facilitate one to differentiate between "rmdir" the Unix command and "rmdir" the Unix system-call. One can say that "rmdir" the Unix command is an instance of topic type *Commands*, while "rmdir" the Unix system-call is an instance of *System_Functions*.
In the standard topic map model, ambiguous topics are differentiated based on subject identifiers. Each subject identifier is a string, which is globally unique for each topic. A subject-identifier facilitates a topic map system to discover whether two or more topics residing in different topic maps represent the same subject. This feature is used for merging different topic maps and aggregating information from various domains.

Cfengine implements a simplified mechanism for identifying ambiguous topics. The cfengine classes in the topic map act as the disambiguation mechanism for topics with the same name existing in two different contexts. e.g. here is an example of identifying the topic-types using the cfengine classes.

```
topics:

Commands::

"rmdir"
association => a("is command of", "Unix", "has command");


System_Function::

"rmdir"
association => a("is system call in", "Unix", "has system function");
```

The relationship between topic type and topic is a class-instance relationship. The second binary association is labeled as superclass-subclass association.

3.2.3 Association

The most distinctive part of the topic map model is modeling of relationships between topics. Topic maps allow one to assert any kind of association between topics. An association establishes a relationship between two or more topics. Association between topics can be grouped according to their type. Topics that have same relationship to a given topic are placed into the same association group type. This feature enhances the expressive power of topic map model, making possible navigating huge amount of information. Association type gives information about the nature of relationships between top-

ics. Without this piece of information we do not know the reason the topics are connected to each other. As example, the association type between the ”slogans” topic and ”Mark” topic is of type *Ownership*. The association can be view in either direction. This means that we can say ”Mark” *owns* the (machine) ”slogans”, or ”slogans” *is_owned_by* ”Mark”. Each associated topic plays a role in the association. The association role itself is a topic. So, in the *Ownership* association ”Mark” plays the role of the *owner* , whereas the machine ”slogans” plays the role of *ownee* . An association might be assigned to a given scope, constraining the association to be valid only within the assigned scope.

3.2.4 Occurrences

Occurrences are the resources that are being pointed to. An occurrence might be anything that can be referred to. e.g. database entries, documents, diagrams, description, etc that are somehow relevant to the topic. There are two types of occurrences, external and internal occurrences. The external occurrences reside outside the ontology itself, but they are accessed to via a Uniform Resource Locater (URL). It corresponds to a page reference in a book index. Internal occurrences reside inside the ontology, and usually they provide a short description of topics. Occurrences are typed, allowing us to distinguish resources. When a user visits a topic, and wants more information about the topic, the user will not get only a set of links, but also what makes each link important. In our CMDB, occurrences such as cpuCount, cpuSpeed, etc are of type MachineResources.

3.2.5 Symbiosis between Promise Theory and Topic Map models

The topic map model provides a framework to address information overload. It does not have an operational roadmap for usage. Users themselves design the structural organization of information. The same approach is implemented by Cfengine, which also provides a framework without a roadmap [12]. The topic maps' ability to connect concepts together independently of hierarchy, and the strength of Promise Theory to represent these relationships clearly and simply would allow us to unify various aspects of configuration management. Topic maps represent the knowledge declared in a set of promises. While promises describe how to design a topic map, its configuration and maintance [17]. This is an example of encoding topic maps using Cfengine's topic map promises.

```
topics:

machine::


"slogans"
association => a("promises to deliver", "Apache httpd", "is promised by");
```

```
occurrences:

slogans::

"http://webmin.com/slogans_how_to_use.html"
represents => ("Planning");
```

3.2.6 Hierarchy and topic maps

One of the unique features of topic maps is that it is independent of hierarchical structure. This means that the topic maps can draw semantic links between topics residing in different topic-types, see Figure 5.4. This capability distinguishes it from the other data models we have discussed. The topic map contains several overlapping semantic links between topic types. It retrieves the concepts residing on databases and describes and links them together in an semantic manner. A single artifact can by pointed to one ore more topics through topic occurrences. The topic map model is distinguished by other information models by existing as a separate layer from the resources, see Figure 3.1.

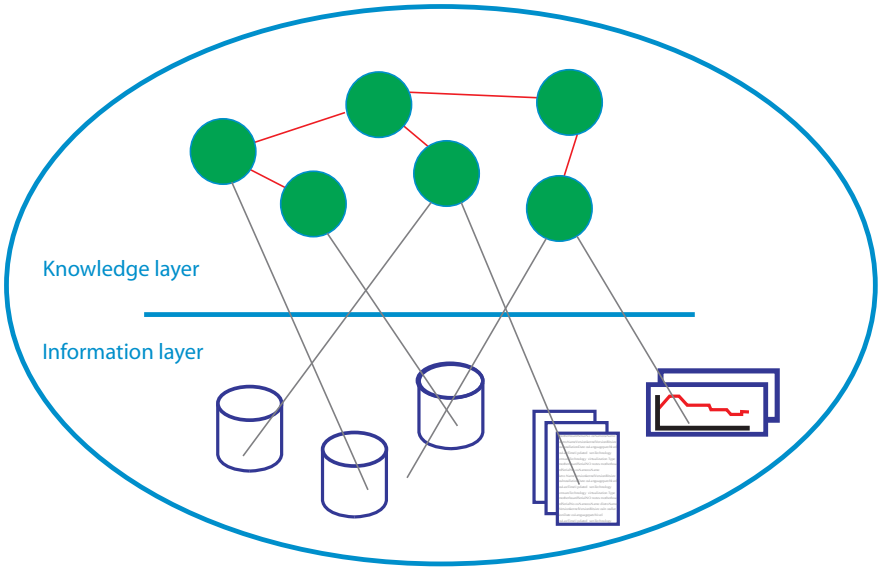

Figure 3.1: Topic maps: Schematic representation of topic maps' features

In an entity-relation model, the connection between objects is dependent on so called primary and foreign keys. An object can not connect an another object if they are not connected through primary-foreign key pairs. Therefore, locating of a particular object has to go through inter-connected objects.
It is costly also in terms of performance. Meanwhile, in an object-oriented model, a class recognizes only its parent class, it does not have any knowledge about other peer classes residing on other side of branches.

In CIM model, in order to connect two different objects we have to create an association class with two references pointing to each of them, see Figure 2.1. An object usually has more than one association, so that we have to introduce for each association a new association class.

3.2.7 Extracting data from relational databases and topic maps

In entity relation databases, end users must know in which databases, the information is stored. e.g. when they query for the desired information, they write an SQL query within which they connect tables based on primary-foreign key pairs. If there is any match it will return a list of records containing detailed information. The end user is supposed to know the database schema. In the topic map model there is no need to know in which table (database) the information is located. The end users do not need to know the low level structure of database model. They get a high-level view of information. The topic map model, in addition to locating the right information, provides also related information, see Figure 5.4. It is more convenient to understand a particular subject if we can relate it to other subjects.

3.2.8 Navigation

Topic maps are powerful in finding information. The information stored in the database schema will be extracted automatically every time we visit a different topic. Topic maps provide two mechanisms for locating information, through searching and associative navigation. Associative navigation yields multitude redundant paths to navigate information. This feature of topic maps does not restrict us to navigate information only through strict predefined paths. In large topic maps it is easier for users to locate requested information through searching mechanism, than through navigating around the topic maps.

Nevertheless, they might find associative navigation useful in enhancing of possibility of finding the information. Meanwhile, relational databases have poor navigation capabilities. They were modeled to use content-based associative access based on SQL. Navigation can occur only based on movement between individual tuples.

3.2.9 Labeling of topics and synthetic keys

In relational databases, it is trivial to create and use synthetic keys for records. The synthetic key for ”operating system” and its version would be:

```
idinstall|os|version
```

and, then the ”idinstall” is substituted for concatenation ”os_version”.

”machine” $\rightarrow$ ”idinstall” (where ”idinstall” determines ”operating system” and its ”version”).

Thus, it easy to generate and use synthetic keys in relational databases.

We have to be careful when we define label names for topics. Topics, when grouped in a class, or split apart, represent something or refer to something on their own. e.g. the topic ”redhat enterprise linux 5” has relations with topics:

”redhat enterprise linux 5” *has_vendor* ”redhat”
”redhat enterprise linux 5” *is_a_distribution_of* ”linux”
”redhat enterprise linux 5” *is_a_variant_of* ”redhat enterprise linux”

But there are more subrelations than that. e.g.
”redhat enterprise linux” *is_a_variant_of* ”redhat linux”
”redhat linux” *is_a_distribution_of* ”redhat linux”

However, not all sub-relationships in topic maps make sense, e.g.
”redhat enterprise linux” *has_version* ”5”
The latter does not make sense, because ”5” does not stand for any topics.
It is not always clear how to break up a break up a complex topic into its reasonable subtopics.

Here is an another example,
”windows XP” *is_a_version_of* ”windows”
”windows Vista” *is_a_version_of* ”windows”
”windows” *employs* ”windows registry”
The latter implies that windows XP and windows Vista have registers.

Meanwhile, we cannot say things like:
”XP” *is_a_version_of* ”windows”
because XP, on its own, without the Microsoft qualification, is not a topic by itself.

3.2.10 Storing of topic maps

Topic maps might become complex, they might contain myriad topics and associations. This size of topic maps will impact performance. Therefore, it is important to select the right repository approach. Certainly, storing topic maps in a file is not the right approach, since the entire topic map will then have to be allocated in memory. Topic maps should be stored in relational databases, because only the requested topic map portion will be read in memory.

3.2.11 Strengths of topic maps model

According to our assessment the key characteristics of topic map include:

- *Ability to link topics to each other in a semantic fashion*
- *Building Metadata*

- *Linking heterogeneous information in one topic*

3.2.12 Ability to link topics to each other in a semantic fashion

In topic maps, information about resources is conveyed through topics, and occurrences. Associations add semantic relationships between topics.

3.2.13 Building Metadata

The topic map model is simple, but powerful. With topic map model we can create metadata and ontology structure with simple means. *Metadata* is literally "data about other data". In a topic map metadata resides outside of real data, describing the characteristics of data. Metadata is important especially when we need to locate useful information from a large amount of information. The topic map subject centric approach provides us the notion of representing resources as topics, and occurrences to gather notions of artifacts residing in different repositories.

3.2.14 Linking heterogeneous information in one topic

Topic maps provide a very flexible and dynamic approach to associating knowledge objects with each other, regardless of their location, and without needing a model for integration. This phenomenon of bringing different artifacts stored in different locations or repositories is known as *co-location*. This capability is unique for topic maps. There is no need to centralize the entire information in a single database, or process it.

3.2.15 Limitations of Topic maps

An association type, within local relationships indicates the importance or criticality of a subject. e.g. "Apache httpd" *depends on* "openssl". This means that "openssl" has to be configured for "Apache httpd" to run. Meanwhile, in large scale, the standard topic model only support the topic relationships and topic navigation, and does not reflect the importance of topics.

3.3 Short introduction to entity relationship modeling

Entity relationship model is based on strong mathematical foundations [13]. It describes relationships in two levels:

- *relationships between entities that are tables consisting of records*
- *relationships between attributes within a record*

The entity-relation model has dominated the design of database systems for more than 30 years. One advantage (compared to object-oriented, and network models) is that it is simpler to understand, and consequently, easier to design database models. The entity-relation model abstraction fits with

mathematics, but it is not convenient for human. It conveys information in an unmanageable manner (as a list of records containing detailed information). Relation models are easy to manage, when they relatively consist of a small number of entities. When relation model is enriched with additional capabilities, its complexity increases with N plus two (N+2), making the model unmanageable. This phenomenon is observed also during designing of our CMDB model.

3.3.1 Normalization versus Denormalization

The entity relation model spreads information across multiple relations. The aim of normalization is to remove redundant data. This is very critical to transaction processing, because otherwise, transactions can violate integrity constraints by contradicting data in one place with data in another place.

Therefore, the entity relation model has been used for designing databases for Online Transaction Processing (OLTP). Normalization enabled by entity-relation modeling. The purpose of normalization is to organize information into more reliable structures, mitigating update anomalies, so that data consistency is retained. Despite the fact that normalization addresses a critical issue for many database implementations, it also introduces three major drawbacks:

- *The model becomes more complex (the number of relations increases).*
- *New relations often do not match objects in "real world", leading to fragmentation of real objects into multiple relations.*
- *The system performance decreases significantly (because of join operation).*

Typical fully normalized relational databases can read a few hundred records per second [14]. This poor database performance occurs because of join dependencies that exists in the model. A disadvantage of entity relation model is join dependencies. In addition to affecting database performance, it makes it more difficult for end users to understand and being able to use database schema.

The inverse process of normalization is denormalization. It is used in many strategic database implementation to boost system performance, and reduce the complexity of database schema. Denormalization techniques are widely applied in data warehousing implementations for data mining transaction. The aim of the data warehouse is to be more competitive through analyzing enterprise data for taking decisions. Typical denormalized relational databases can perform aggregation at 10.000 records per second [14]. To reduce complexity and increase performance we have excluded the application of normalization in our CMDM model. We do not need it because our CMDB is not the authoritative source of data. Our CMDB will function as a cache.

3.3.2 Lack of semantic capabilities

The entity relation model provides only one construct, for modeling objects and relationships between these objects. e.g. in order to model relationship of

type many-to-many between *Machine* and *Person* objects, we have to create three entities, two of them represent the objects, while one represents the relationship between these objects. The entity relation model does not provide a facility to differentiate entities that represent the objects and entities representing the relationships, or differentiate between different types of relationship that exist between entities. Technically, the entity relation model labels relationships between objects in a semantic manner, such as *owns* , *administrates* etc, whereas in the relational databases this powerful capability cannot be achieved.

3.3.3 Similarities and differences of CMDB and Data Warehouse

The term Data Warehouse originally was invented by Bill Inmon, who described it the following way:

"A warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision making process ".

- **Subject-oriented:** Data provides information about enterprise's business processes.
- **Integrated:** Data is collected into a data warehouse from various sources and structured to a common format.
- **Time-variant:** Data in a data warehouse is accompanied by a time parameter. It is critical piece of information for Data Warehouse.
- **Non-volatile:** New data is added into a data warehouse and remains for eternity. This allows enterprises to get a consistent view of the business.

Our CMDB and Data Warehouse are similar in the sense that they address common goals.

- **Competitive advantages**
- **Data Warehouse:** enterprises obtain previously unavailable, unknown information, e.g. information on customers, trends, and demand.
- **CMDB:** enterprises will be able to deliver IT services at lower cost.

  Increased efficiency/productivity of enterprise decision-makers
- **Data Warehouse:** decision-makers can conduct substantive, accurate, consistent analysis.
- **CMDB:** IT management can drive process automation.

The most common schema of Data Warehouse is the star schema. The star schema is a logical structure, which has a fact relation in the center, congaing factual data, and is connected to dimension relations containing reference data. The dimension relations are denormalized. The fact relation contains the data, dimension relations contain description of data. In star schema, the fact relation and dimension relations are connected through primary-foreign key relationships.

Despite the fact that our CMDB behave like a Data Warehouse they are also characterized as being two different technologies.

3.3.4 Characteristics of standard Data Warehouses

Data Warehouse depends on online transaction processing (OLTP): Data Warehouse fetches constantly a copy of transaction data from OLTP.

Long term project: Enterprises can retrieve the benefits of investments in Data Warehousing implementation after at least five years.

Incompatible sources: Information has to be collected from a multitude of incompatible sources into a common format, which provides a consistent picture of the enterprise.

Data cleaning: Different programmers have designed OLTPs with different criteria, therefore "data cleaning" has to occur before transaction data ends up in the Data warehouse.

Here we list the characteristics of our CMDB compared to Data Warehouses

- *Population of CMDB with data takes place dynamically. The process will be completed within a reasonably short period of time.*
- *There is no need to perform "data cleaning".*
- *CMDB serves as a cache and will be recreated on demand. (It is like a search-engine)*

3.3.5 Indexing of attributes

Relational databases read records from physical tables in a sequential manner. This approach has impact on databases' performance, especially in cases where there is a tremendous amount of information stored in a databases.

Searching for data through primary keys will actually not render significant performance problems, because tables are by default indexed by their primary keys. However, in many cases, it is not convenient to search for data via primary keys, since they are complex. Meanwhile searching for data through common attributes is much easier, but they are not indexed by default and consequently the respond time can be much longer.

There are situations where we need to search for occurrences via common attributes. e.g. we might query the CMDB to find all machines that have Pentium 3 as processor. Therefore, to enhance the response time, we might index attributes that we are supposed to search through.

Meanwhile, indexing of common attributes is not without cost. It is required to update these indices along with records. Therefore, indexing every single attribute is not a good idea. The adequate approach is to identify attributes that most likely are going to be used for searching for occurrences, and then index only those.

E.g. for the MachineResources database, we suggest indexing only the hostname attribute, whereas for Architecture database we would index the following attributes: motherboardArchitecture, cpuModel, hddModel, ramModel.

3.3.6 Fundamental limitations of entity relation model

Entity relation models are generally good for storing simple ”flat” data which fits to entity relation model. This is also the main limitation of entity relation model. It does not handle documents, diagrams, etc. The entity relationship model forces data to reside in a rigid information model. In a business environment the need for adding additional capabilities is increasing. This means it is required to update the rigid schema. This process is very difficult if it is required to redesign schema frequently. Changes made on schema has to be reflected to application interface.

Even though enterprises are inclined and are capable of adjusting the database schema in order to meet new requirements, they are unable to implement these modifications, because redesigning the information systems is costly in terms of time and money (Taylor, 1992).

Data interchange is a problem also for data warehouse technologies. Before populating of data warehouse from relational databases, ”data cleaning” must occur. Programmers are supposed to observe the information structure on all relational databases, then write program that will give all information same structure, before they end up in data warehouse database.

In relational database, information can be stored only after the schema is designed. This means that information structure that is not supported by database schema cannot be added, unless the updating of schema take place. e.g. before we add information about operating system, its physical table must exist. Meanwhile, in topic maps we can update their schema while they are in operation. Therefore, minor database schema updates can be avoided, through updating toic maps.

Entity Relation model cannot handle subjects with the same name. It is an argument that Entity Relation model is not powerful to represent every "real object". On other side topic maps can denote alternative names for subjects.

3.3.7 Determination of primary keys and subject identifiers

In entity relation model, the determination of primary keys for entities has to go through a process. Overall primary keys should be unique within an entity and functional. Selecting of a non-key as a key, leads to incompleteness (two updates with the same key overwrite one another). In our first CMDB draft, for machine entity we selected as primary key just a string labeled *machineID*. Later on, we raised the question how do we control it, since CMDB has to be populated with data dynamically. Thus, even though the selected key was unique, it was not functional. Therefore, we changed it to *motherboardSerialNo* , because this value, in addition to being unique for a particular machine, could be fetched automatically by a cfengine agent. Meanwhile, in topic maps, topics can have an identifier, which is labeled as subject-identifier. It can be any string, but for convenience it is standard to write it as an http address.

Chapter 4

Methodology

We aim then, to look for a new model of CMDB with minimal complexity and use Topic Maps' semantic model to link items and classifiers together in a more flexible way. In the absence of a recognized methodology, we proceed by trial and error, gradually exposing errors and flaws in the modeling.

4.1 Development of CMDB model

During the process of development of a logical database (CMDB) model, we have sketched different versions of CMDB models. The requirement of designing a logical database model, which provides certain distinctive characteristics compared to existing models, led to development of various drafts for CMDB models. The first three CMDB data models were designed based on principles Entity Relation model (ER) and Topic maps. The forth and fifth drafts were designed based on the principles of boxes and Topic maps.

Development of logical database model has gone through different stages. Based on their structure and capabilities, the drafts for a logical database model are listed below.

- *Single CMDB database model*
- *Information types, ER-entities, and Topic map*
- *The three databases CMDB model*
- *Introduction of box concept*
- *Application of Dunbar number*
- *Introduction of change concept*

4.1.1 Single CMDB database model

The simplest view one can have of a CMDB is that it is a database that stores a huge amount of information. It is up to designers to organize this data logically inside the database. Bearing this fact in mind, we started to model our

logical database using standard ER-model. Each object type was represented as an entity and its attributes. Drawing of relationships between entities almost always results in introduction of *inter-entities* (because of many-to-many relationship). Every time we added a new object, this corresponded to adding at least two entities to our CMDB model. As an example, when we added *software* entity to our model, and we drew its relationship to the *machine* entity, we were forced to introduce an another entity between them, in order to depict what applications were installed on various machines. It means that our CMDB model was growing with *N plus 2* , every time we added a new element. N is number of entities

The single CMDB (first draft of CMDB) could be populated automatically with configuration information. However, there is a major drawback. Population of the single CMDB must occur in a particular order. So, before a machine reports services it promises to *providedServices* entity, it has first to report to machine and service entities. Otherwise the reporting attempt will fail if foreign keys do not match its related primary keys in machine and service entities. To make sure that reporting process would be successful, inter-entities, such as *providedServices* have to populated last. This serial dependency affects also collecting of information. One prefers to report data to the logical database in proportion to the rate at which data changes. Data that remain static or change slowly can be communicated infrequently, such as a machine's resources or hardware description. Whereas data that might change often should be communicated more often, such as service status. This is not possible in single CMDB, because of ER-relationships between entities.

The strict application of the Entity Relation model's rules made our logical database model quite complex in breadth and depth. Every time we added new elements to model, it was reflected in additional complexity. Inter-entities contributed to complexity in both breadth and depth. While information's atomic properties affected complexity in depth.

4.1.2 Information types, ER-entities, and Topic map

A datacenter has several types of information, stored in different places. Accordingly, we raised the question,

What types of information are appropriate to store in relation database, and what types in topic map ?

Raising this question has facilitated us to highlight some drawbacks of our logical database model. In the previous logical database draft, electronic documents, such as contracts, reports, etc were modeled based on principles of the ER-model. This was also one of the reasons that those models became complicated. For some of document types, such as contracts, we were compelled to add two entities to logical database. Nevertheless, this was not the end. Information was supposed to be extracted from original source through special programs and put into the databases. When the user read one of these files an

another program read the database and created the required contents as a file. Furthermore, we could not escape entering some data manually into the CMDB, such as information on third parties, suppliers etc. This form of handling document files is offered by CMDB vendors today. The ER-model is very poor model for handling document files, it is required to process information residing on documents at various stages.

According to our analysis, the topic map is adequate for storing small amount of data, and pointing to electronic documents. A topic map using semantic associations can link any topic to any type of data structure, including files, and there is no need to process any information. Here lies the strength of the topic map. It is created for managing the meaning of information, rather than processing of information. Topic maps allow us to make different interpretations about same information. If we take, for example, database tables ”Machine” and ”Person”, we can link concepts according to our needs. machine: ”slogans” *is owned by* ”Mark” or machines: ”nexus”, ”dax”, ”vax” *are maintained by* ”Danny”. This flexibility does not exist in the standard ER model. Recall from our single database model. Between *Machine* and *Person* entities, we introduced the *Maintainer* entity. It represents a limitation as to what type of information we can store on it.

Meanwhile, ER-entities are adequate for storing huge amount of information discovered automatically or manually. We type data manually into *People* and *Location*, therefore we use an ER-model, since it is more reliable to store manual entered data on ER-entities. These findings are applied in designing our third logical database draft.

4.1.3 The three databases CMDB model

Virtualization has brought changes to datacenters. It has changed the way we manage resources. Virtualization detection and monitoring capabilities were not supported by single database CMDB.

Adding virtualization capabilities to our logical database model will enrich it. We had mainly two purposes for supporting virtualization. We wanted to track deployment of running virtual machines and monitor certain parameters of them, such as utilization of resources. This type of information would help us to draw resource management decisions. e.g. resources of a machine that are being used heavily would be excluded from deployment of new virtual machines. Because it would degrade the performance of the machine.

Extension of our logical database with virtualization support, led to redesigning of single CMDB model. Now, we store data that changes at different rates. Precisely, we reflected this new requirement in the new logical database draft. According to this criterion, we have three types of information.

- *manual changes*
- *slowly changing (automatically discovered)*

- *quickly changing (automatically discovered)*

The idea was to store each type of information in a separate database. Consequently, the new logical database model consisted of three databases. The *Person* database stored information on employees. This information changes very rarely. A second slowly changing database is intended to store hardware and software configuration information. The contents of this slow changing database change only when we adjust hardware or install patches and security updates. Configuration information is stored on standard ER database. It consisted of 10 entities linked together through ER relationships. Configuration information is considered to change slowly. The *virtualization* database apart from storing information where virtual machines are deployed and resources they have been given, it stores also information on resource utilization.

4.1.4 Redesigning of location entity

One of the questions our logical database answers is how to display physical location of machines. In the single CMDB model, location were named straightforwardly, e.g. room100. In the new approach, we mirror the room structure of a datacenter. Machines in datacenter's room are arranged according to an order, where first is assigned a row, then a rad number, and finally a position. Adding the building and room labels, the final physical location address of a machine on CMDB would be, e.g. headquarter_room100_4_3_5. This labeling approach helps us to find machines rapidly in a datacenter. But it does not work for laptops, since they do not have a fixed location. Redesigning of the identification attribute changed relationships between *Location* and *Machine* entities from one-to-many, to one-to-one.

We tried to introduce a new approach in organizing information in the logical database model. The configuration information database inherited all drawbacks from its predecessor. We needed a new approach to address the complexity and dependencies of logical database model. So, the question was:

How do we overcome the standard entity relationships and simultaneously retain the logical database connectivity ?

4.2 Introduction of boxes

Dependency makes a system vulnerable to failure, difficult or perhaps impossible to scale, or difficult to extend afterwards, since those might be in conflict with existing components. A dependent system does not have broad application to different business types. It is applicable only to particular systems within the designed frames. To overcome dependencies between entities and reduce complexity we have introduced the *box concept*.

Boxes are built using some of the principles of ER-model, but do not mimic the entire structure of ER-model. The cornerstones of ER-model, ER-relationships

between entities and information's atomic property are not implemented by boxes. The latter, boxes handle in a flexible manner. Consequently, a box is not a complete product of ER-model. It is distinguished by its simplicity. There are no ER-relationship types between boxes. The other attribute that distinguishes a box from an ER-entity is that the box is independent from its related boxes. Meanwhile, in ER-model relationships between entities are dependent on *primary key* and *foreign key* infrastructure. When we enter a record on an entity, the foreign key, must match the primary key that corresponds to its related entity, otherwise the attempt will fail.

Boxes preserve the strengths of the ER-model. More precisely they preserve primary key and information's atomic property when it is appropriate. A primary key is an identifier of a record residing on a box. This identifier is used in order to eliminate data redundancy in our boxes. Elimination of data redundancy is fundamental property of any kind of databases, including logical databases (CMDB). In addition, if the database stores huge amount of information, it will impact system's performance. Boxes are flexible in handling information's atomic property. Information that stands unique will be kept atomic. As an example, the amount of memory a machine has. Whereas, pieces of information that complement each other will be concatenated. As an example, distribution name and distribution version of an operating system yield more information when they have been merged. This method of processing information has an another positive effect on boxes. They will become shorter in length.

4.2.1 Turning entities into boxes

Ahead ourselves we had a complicated and constraint logical database model. It was built upon the strict principles of entity relation model. The issue was how to overcome these hurdles, so that we have a simple model, but rich in capabilities. To avoid treatment of inter-entities (*dependencies*) we added machine's entity identification key to main entities such as Service, Software, and Virtualization. The impact was awesome, no dependencies, no complications, easily understandable and manageable CMDB model. The new formed units, we labeled database boxes. The identification key of Machine database box lies also on Service, Virtulalization and Software boxes. The identification key of the latter was placed in Package database box. Now, we have logical connections between database boxes.

4.2.2 Redesigning of machine box

Introduction of boxes has reduced the breadth and length complexity of logical database model significantly. However, the machine box remained very complex in length. It contained 44 different variables. The machine box stored different types of information, from machine' resources and hardware details to operating system details. To reduce the complexity of the machine box, we started to classify information. We could add an additional hard disk drive

to a machine, without having to know kernel version of operating system. Therefore, we split the machine box into *MachineResources* and *OperatingSystem* boxes.

4.2.3 Handling of dual-boot

Splitting of *Machine* box into *MachineResources* and *OperatingSystem* boxes reduced the complexity of handling a huge amount of information significantly. The *MachineResources* box (as the name implies) is designed to hold mainly resources' metrics for machines. Whereas the *OperatingSystem* box is designed to contain details on operating systems installed on machines. One issue that the *OperatingSystem* box did not address was dual-boot. To address this issue we concatenated motherboard serial number together with operating system name. Concatenation of these two pieces of information yields unique identification for each of operating system instances installed on a machine.

4.2.4 Tracking of machines and virtual machines

During the process of development of CMDB model, we considered that it is a good idea to add *timeStamp* attribute on boxes, especially on Machine and Virtual machine database boxes. Every time there is an update in the logical database, the exact updating time will be recorded on timeStamp field. This new capability would allow us to track machines and virtual machines that were reporting to CMDB and those that have stopped reporting. If a server is not accessible through the network, we can swiftly find out if there is a connection failure just by referring to the timeStamp attribute. It would show us the last time the server reported to the logical database. If it is up to date, then there might be some misconfiguration, or the firewall prevents reaching the server.

4.2.5 Adding of attribute notes into our CMDB model

One of the issues that CMDB authors highlight is determining the scope and level of CIs. The flood of information introduces delays, and might result in a failure to achieve specified objectives. In the logical database we store configuration information at the level at which it is manageable. Positively, we store data that is needed to support the enterprise's business services.

The *notes* attribute is added into CMDB model to support various types of information on entities. This makes our CMDB model extensible introducing the second dimension on database boxes. The common attributes will hold the most important configuration information of an entity, while the *notes* field will contain everything else that cfengine agent discovers. Occurrences of the *notes* field will be linked back to the topic map layer, forming a fully embed topic map index.

4.2.6 Operating systems and platforms

Operating systems can not run on all hardware architectures. e.g. Solaris is constrained to run on SPARC, x86, x86-64 platforms. It is important to provide this information to end users. This type of information will be stored on topic maps, due to the fact that this information is generic for all machine architectures and is easier to refer. This is manually entered information, therefore storing this type of information in *OperatingSystem* box is not adequate, because it will be removed when CMDB recreation occurs.

4.2.7 Removal of monitoring data from VM Box

In previous draft, the *VirtualMachine* box was designed to store static data and monitoring data. In fact, this database box would store two types of information, more precisely static data (as the name implies this data changes slowly), and monitoring data, (which changes quite frequently). Monitoring data is plotted on timeseries graphs, and in order to obtain approximately real data, we have to measure the environment frequently enough.

This is certainly not an appropriate solution for centralizing monitoring data on a database. In order to obtain approximately real timeseries data, we have to update the database at the rate that timeseries data changes. This is very difficult, because updating of a centralized databases is costly. In addition, we have to repeat this procedure for hundreds of machines. Therefore, it was natural to exclude monitoring variables from the *VirtualMachine* box. Separation of static data and monitoring data is also fully in accordance with principles of our CMDB model. According to our CMDB model different types of data should be stored on different boxes. Monitoring data will be stored on reports locally in each machine, and handled through topic maps.

In the new design, the virtual machine box stores only static data, more precisely cpu, disk and memory resources that virtual machines have received. In addition, the virtual box keeps track of running virtual machines and links each om them to its physical machine. Redesigning of VM box reduced the complexity on length. From 25 elements it contained in previous draft, now it contains only 13 elements.

4.2.8 Powering of virtual machine and operating system boxes

Despite the fact that we introduced significant changes to VM box, it still lacks some important capabilities. *VirtualMachine* box did not support discovery of inactive virtual machines. This is a weakness of the *VirtualMachine* box, because the datacenters might retain inactive virtual machines, which they activate on demand. The new version of virtual machine will keep track of virtual machines in a datacenter regardless of their running state. The updated VM box provides information where the virtual machines are deployed and how long they have been in operation. The predecessor version of VM box did not support discovery of inactive virtual machines.

At the planning stage of designing the*VirtualMachine* box, we had as a purpose of tracking only virtual machines from Xen-technology. Meanwhile, considering the fact that vmWare has got a broad deployment in datacenters, we decided that our model should support vmWare virtual machines as well. To indicate the kind of virtual technology for virtual machines, we added the attribute *vmBranchTechnolgy* on *VirtualMachine* box.

In predecessor version of *OperatingSystem* box only paravirtualization type is supported. Due to the fact that our initial approach supported only Xen virtualization technology. Xen technology is known to run virtual machines on paravirtualization mode. Since we have expanded *VirtualMachine* box to support vmWare technology, and it runs virtual machines in full virtualiaztion mode, we had to add this information on *OperatingSystem box. To store information on virtualization type implemented, we placed the* virtualizationType *on* OperatingSystem box.

4.2.9 Adding of anomaly detection capability

Observing of machines and virtual machines running status through the timeStamp attribute was a decent feature. We could swiftly find out whether machines were reporting to the logical database. Nevertheless, this was a straightforward approach, we either know that a machine is reporting or not reporting. It does not contain any evidence for machine's anomalies. This means that machines report to logical database, but not at regular times. Thus, we have limited information, since we do not know if there is something delaying machines from reporting to logical database. Therefore, we were interested in capturing this evidence, due to the fact that it facilitates us to heal anomalies, so that machine's resources utilizes in more efficient manner. We replaced *timeStamp* with *deltaT* attribute. We specify explicitly the timeframe that machines or virtual machines should report to logical database. If the reported time exceeds the timeframe, it means that the machine suffers from anomalies. This would followed with an alert issued by *Cfengine* . This capability is unique to our logical database model. It is not provided by other CMDB vendors.

4.2.10 Creation of architecture box

Our model organizes information into different boxes, based on the type of information. By observing the current draft of machine resource box we noted that two pieces of information were not supposed to belong to this box according to principles of our model. The attribute *formfactor* shows the type of machine, e.g. laptop, desktop, or mainframe, whereas the attribute *vendorProduct* shows the manufacturer of machine. Meanwhile, we as users most of the time are interested in knowing the capacities of a server, and it may not be useful knowing whether the machine is produced by HP or Dell. Therefore, these attributes along with serial numbers of machine's components such as cpu, disk and memory are placed in a new box, named Architecture.

4.2.11 Redesigning of machine resources and architecture boxes

After completing the logical database model, we concluded that the machine resource box should store only resources' metric and other information that directly determine the performance of machines. Meanwhile, attributes that are stored such as cpu model, display model, etc were moved to Architecture box. This was the right thing to do, since these types of information naturally reside on Architecture box. So we have made a clear-cut distinction between resources' capacities and their hardware description. We excluded also warrentyExpirationDate attribute from the machine resource box, because we access this information from contract documents through topic map occurrences.

4.3 Application of Dunbar number to boxes

The introduction of *box concept* added some unique capabilities to our logical database model. Boxes address some well known issues, such as, balance between depth and breadth, make the CMDB model extensible, and accounting for human constraints in processing huge volumes of information. The latter means that a box offers only what we are interested in. If we are in a *MachineResources* box, it displays only the machine's core hardware resources.

The size of boxes is determined based on the findings of British anthropologist Robin Dunbar, who writes [7]:

"Humans have a cognitive limit of approximately 150 on the average number of individuals with whom coherent personal relationships could be maintained" .

His findings are based on the size of neocortex, the bigger the neocortex, the bigger social group a species can maintain. According to Dunbar, attempts to cross the threshold unavoidably result in social group fission. Translating Dunbar's principle to processing of information, it means that humans lose oversight capability over huge volumes of information. What makes Dunbar's findings interesting to our model is that he has divided the number 150 into multiple groups based on the importance of relationships. The core social grouping consists of 3-5 individuals. It is the group from which a person would seek advice or help when he is in crisis. The following group is referred as a sympathy group. It has 12-20 individuals, with whom we retain special ties. The third group is referred to as bands. These groupings are formed by 30-50 individuals and are unstable. We rendered Dunbar's findings into boxes' size to be within the core and sympathy groups. Findings of Dunbar are of particular importance, since they allowed us to model boxes' size in proportion with human's limitations. This would facilitate human to process information effectively and consequently take the right decision. Statistics shows that humans ignore messages in cases when they are confronted with huge amount of information.

4.3.1 What does Dunbar's finding mean for the model ?

This means that the model should generally remain within the predefined frames, in order that human can understand and apply it efficiently. This means that adding new elements into the model continuously will make the model at some point complex to understand and maintain and even useless. In order to be rich in features and simultaneously simple, the model should discuss mainly important aspects of the object in question.

4.4 Introduction of change concept

It was necessary to introduce the change concept into the database boxes, since they store information that changes at different rates. This new approach classifies information stored in database boxes further. This would allow us to communicate data to database boxes at proportion with the rate at which data changes.

4.4.1 Advancing capabilities of OS box

The previous logical database drafts supported only descriptive information of operating systems installed on machines. The *OperatingSystem* box was fairly narrow. It stored information such as the name of operating system, its version, the distributor etc. The power of operating system box started to enlarge when we added virtualization and security capabilities. Regarding security issues, at the beginning the *OperatingSystem* box provided information on service pack version or more generally patch level, and last time security patches were installed on machines. Keeping this software updated is critical in protection against malicious codes. Meanwhile, to enhance the strength of operating system box further, we add also firewall capability. The *OperatingSystem* box was supposed to provide information on firewall status.

The idea was to strengthen the operating system box, so it offers a security package. Reviewing of principles of information model led to defining more accurate criteria for organising data on boxes. According to new principles, slowly changing data and monitoring data should be kept separate. Consequently, we moved firewall capability to a *Service* box.

4.4.2 Properties of boxes

A box represents a certain view of an object. A box is not a container of "everything". It is the storing space for ranked information of a projection. This means that boxes contain the amount of information that human brain can cope with. Inside the boxes can be stored automatically discovered configuration information or manually entered information. It is not recommended to store electronic documents. In our CMDB model documents are handled through the topic map model.

Boxes are manageable, and provide powerful capabilities. Boxes provides only information that users need, they would not provide information that

users are not interested in. If a user wants to view capacities of a server, he would not be overwhelmed with lateral information such as displaying the last time the operating system was updated or a distribution's version. Boxes handle large number of occurrences. Boxes allow us to balance the complexity of our model in regard to breadth and depth. Population and updating of the logical database is independent of any predefined sequence.

4.5 Centralization

The logical database is a centralized facility. We draw a line on what type of information will be centralized, and what type of information will be held locally on the machines. Static data and slow changing data we register to logical database. Meanwhile frequently changing data (monitoring data) we don't centralize. It is too costly to centralize monitoring data to a database. In addition, a datacenter has thousands of machines. Monitoring data is stored on local machines. We retrieve this data from the local machines through topic map occurrences on demand, see Figure. Monitoring data that is not needed will not be created or maintained, since it consumes resources. When we need monitoring data we request the local machine to generate it. This approach requires that we have to wait until the requested monitoring data is generated.

4.6 Promise theory

Promise theory was invented to model the behaviors of agents [15]. The fundamental assumption of promise theory is that objects participating in exchanging information or behaviors are truly autonomous agents. They decide their own behavior, external factors cannot drive them to other behaviors. Interaction between the agents are determined based on voluntary cooperation. The convergence feature is characteristic to promise theory. Convergence is an approach for achieving steady state behavior in a stochastic environment.

Defination 4 *(Promises) A promise is a directed link that consists of two autonomous agents and a promise body, expressed* $a_1 \xrightarrow{b} a_2$. *Agent a1 (promiser) offers a behavior, while agent a2 (promisee) utilizes it. The promise body b describes the nature and constrain of the promise. A promise is expected to be kept and verified.*

There must be interactions between agents for successful accomplishment of exchange of promises. Promises made by agents are of two types, promises to offer a behavior, or provide something (expressed $a_1 \xrightarrow{+b} a_2$), and promises to use the promiser's behavior or accept something (expressed $a_1 \xrightarrow{-b} a_2$).

The information model, we developed for logical databases is compatible with promise theory. Each box promises a type of information and the rate at

which information changes. The machine resources box promises to deliver information mainly on machine resources, and that data changes quite slowly. The operating system box promises other promises. It promises to deliver information mainly on operating system and that its data changes a bit faster compared to information promised by machine resource. Promise theory allows us to make a schedule for capturing configuration information. Configuration information that remains static or change slowly would be communicated seldom to logical database. Whereas information that changes often would be communicated frequently to logical database.

4.7 Timescale and ranking of data

One of the distinctive features of our CMDB model compared to existing CMDB solutions, is that it supports *timescale*. It allows us to communicate configuration information to logical database at proportion with the rate at which configuration information changes.

We register the static information to logical database only once, and when we recreate the logical database. Meanwhile, slowly changing data, we communicate to logical database not so often. Monitoring data such as resource utilization, uptime, etc is stored locally in each machine. This type of information is not communicated to the logical database, since it is too costly to centralize monitoring data. We try to communicate information to the logical database only when there is something new. Therefore, we emphasize that *timescale* approach is an efficient way of collecting information.

One of the criteria of our CMDB model in organizing the information is the rate information changes. Information that changes at different rates are placed into two different boxes. This helps us to rank the data by how often they change. Information stored in *Location* and *Person* database boxes does not change, these stay static for (relatively) long times. Description information of hardware remains unchanged during the lifecycle of the machines, therefore it belongs to first rank.

Information stored in the *MachineResources* box changes slowly and is placed on second rank. This type of information needs to be updated generally when we add/remove physical components, virtualize the machine, populate the *MachineResources* box with additional components etc. Information on operating system changes a bit faster compared to machine resources. This type of information is ranked in third place. This type of information changes when we install patch levels, security patches, add Xen virtualization module to kernel, etc.

It is flexible to configure virtual machines. We can configure virtual machines with various parameters, depending on our needs. Consequently, this type of information is supposed to change faster. Information stored on services box needs to be updated often. The *Service* box contains among other information about service availability status. In a datacenter, there are many services running on different machines. Services, such as web, or firewall might go down due to misconfiguration or other issues. To obtain the availability status

of services we are enforced to monitor this data continuously within certain intervals and report their status to the logical database.

4.8 CMDB as cache

Our primary source of data are machines, which means that we can recreate the CMDB any time and no problem emerges. Even though a machine has been disconnected from the network, we would not lose the knowledge about the existence of that machine. Machines reside in policy configuration file as promises. Topics representing the machines that do not point to occurrences of the *MachineResources* database box will be listed, allowing us to locate them swiftly. Our CMDB serves as a temporary cache. We dot backup the cache data, it can be regenerated at any time. The cache data will be used for searching.

4.9 Fault tolerance

Our CMDB is not vulnerable to misconfiguration of machines. In case that a bunch of machines were configured to report to logical database constantly, this not will affect the logical database in terms of its integrity. The logical database will remain consistent and will not grow in size. Only the updating of existing data will take place.

4.10 Polyscopic structuring of information

Polyscopic modeling was developed at the University of Oslo (Dahl et al., 1972, Karabeg 2003a). The polyscopic information model organizes information hierarchically and modularly. We developed principles of our CMDB model independently of the polyscopic model. Meanwhile, observing some of similarities that exist between these models, we have listed principles of poliscopic model. Simultaneously, we have highlight differences that exist between two. The principles of polyscopic information model, quoted from [16]:

- Information has multiple aspects, some of which are subtle or hidden. The most relevant aspects must be provided (horizontal abstraction).
- Information is a function of the way of looking or scope. The scope is coherent if it represents a single level of detail and angle of looking (or intuitively, if it reflects a single viewpoint on the metaphorical mountain). Each information module must be associated with a single coherent scope (coherence).
- The knowledge of the scope is crucial for proper understanding (intuitively, one must know where on the metaphorical mountain one is standing when looking at a piece of information). Each information module must be associated with a clear and correct representation of the scope (orientation).

- The scope determines the view (intuitively, the scope is the control knob given to the reader to switch between views). The reader must given the capacity to select the view by changing the scope (navigation).

One important issue that the polyscopic information model does not address is classification of information. A polyscopic model contains unclassified information. The main concern of the polyscopic model is that the view has to be coherent. As we have discussed in our principles of CMDB model, a view contains a lot of information of different ranking. Consequently, it causes information overload. This issue is not addressed by the polyscopic information model. Apart from this the rate of information within a view is not modeled. Polyscopic model does not give any guide on how projected views should be connected to each other.

Chapter 5

Designing of topic maps for CMDB, and Results

5.1 Principles of our CMDB model

From the previous discussion, we can identify some principles for the design of a CMDB. We may consider a machine projected into different boxes, reflecting the information it offers. The angle of projection determines the contents of a box. Each projection forms a box, which stores only a type of information. So, inside the *MachineResources* box is recorded only the resources of machines. While description information about the hardware is stored in an another box labeled *Architecture*. Projection allows us to classify information into different boxes. Boxes are connected together through the object's identification key. It it a string that is unique for each object. For machine object we have selected motherboard serial number as identification key, since it is unique for each machine. The box concept is accompanied by a *change concept*. It is a characteristic of a box. The change concept allows us to keep related information into separate boxes. Pieces of information such as, the amount of memory card and memory card model is about the same object, precisely memory object. However, the change concept distinguishes them from rate at which they change. The memory card model information is static and is placed on Architecture box, whereas the amount of memory card information might change if we upgrade the machine, therefore this piece of information belongs to an another box. Slowly changing information and static information are placed into two different boxes.

An object might be complex, e.g. the machine is a complex object. A projection yields a view of the object, which itself contains a long list of information of different ranking. e.g the operating system projection among other contains information about shells it supports, filesystem it can process, modules it has mounted, etc. This diversity of information within a view or box render information overload. To remedy information overload of a view, we apply Dunbar's findings. On boxes we store only information, which is important to us. Information that is native is excluded from boxes. This would allow humans to process information in an efficient way.

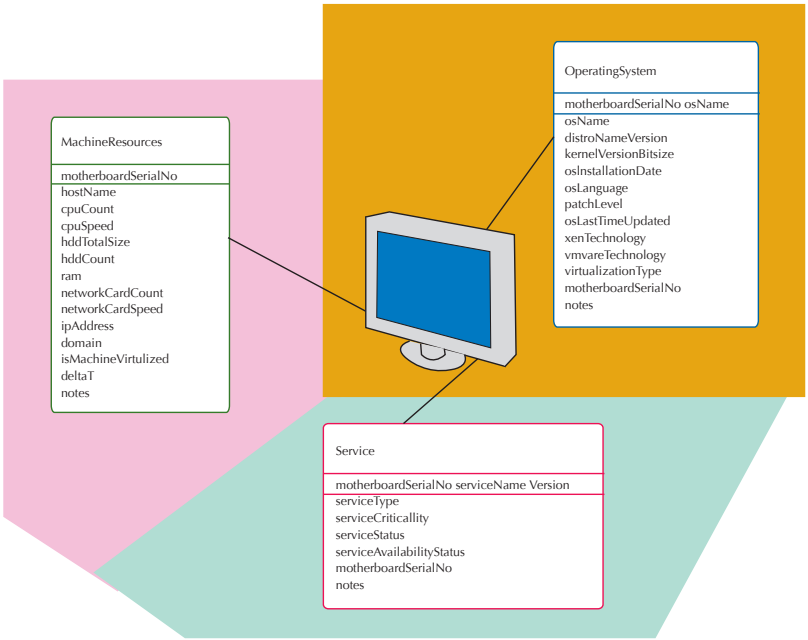

Figure 5.1: The projection metaphor is one of the designing principles of our CMDB model.

In our CMDB model, we introduced the *notes* field in database boxes to avoid the need for redesigning the CMDB model. If there is new thing we want to add to CMDB, we put it in *notes* field. Its contents will be organized in topic map structure. This would allow us to link occurrences back to topic map ontology forming a fully integrated index.

5.2 Mapping of CMDB model to Topic Map model

Our final CMDB model is consisted of nine database boxes. Each database box stores a different type of information. In the topic map model, the approach was to model each database box as a topic-type, see Figure 5.4. We consider this phenomena as a simple relationship between topic maps and database boxes. While, each record of a database box represented an instance of corresponding topic-type. Occurrences of database boxes are a given type.

For designing the topic map ontology for our CMDB we have selected GTM (Graphical Notation for Topic Maps) as a modeling language. As its

name implies, GTM was created specifically for modeling topic map ontologies. GTM provides different node shapes for representing different topic maps constructs. This rich representation of constructs allowed us to design the topic map ontology for our CMDB fast and easy. Foreign topic maps designers can easy understand the structure and elements of our CMDB topic map ontology.

Despite the fact that there have been efforts in enhancing the capabilities of UML including designing ontologies, UML [17] lacks some fundamental abilities such as:

- *representing of relationships such as "is similar to"*
- *representing formal semantics of sets*
- *representing synonyms, antonyms, homonyms, etc.*
- *ability to draw multiple relationships of an entity to each other.*

5.2.1 Topic map ontology of our CMDB model

In our CMDB topic map ontology we represent constructs: topic-types, topic-type associations (association types), role types, name types, occurrence types. The Figure 5.3 points out the topic map's constructs.

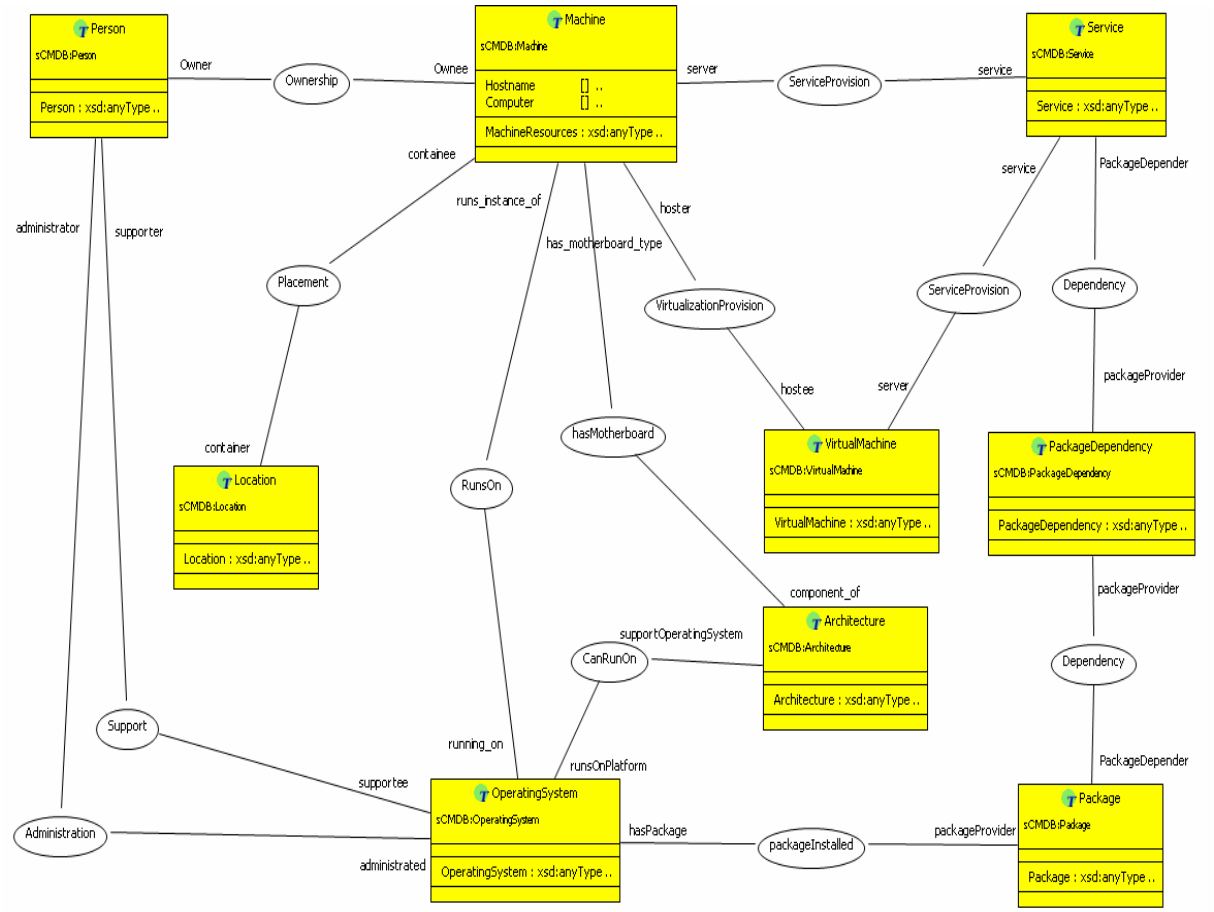

Figure 5.2: Topic map ontology of our CMDB model

The topic-type association indicates that instances (topics) of these topic-types can draw association links of a given association type between each other. Topic maps are a type of semantic network containing topics and associations describing the relationships between subjects. Topic map ontology in the Figure 5.3 precisely indicates the network oriented structure of topic maps. A topic can point to different occurrence types. To keep the figure clear we removed the links between Topic types and occurrence types. But in the test case we indicated the relationships between topics and occurrences, See Figures 6.1, and 6.3

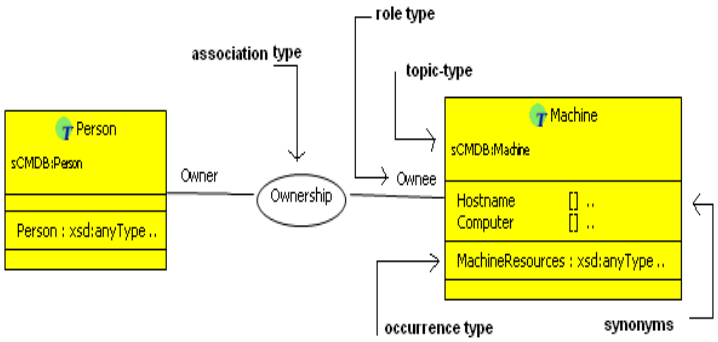

Figure 5.3: Constructs of the topic map ontology of our CMDB model

Figure 5.3 shows that the instances of topic-types Person and Machine establish relationships of type *Ownership*. Instances of the topic-type Person play the role of ”owner”, while the instances of topic-type Machine plays the role of ”ownee”.

5.2.2 Sample

This sample is designed from the topic map ontology of our CMDB model. The MachineResources database box in the topic map model is mapped to a Machine topic-type. It has occurrences of type machineresources, and synonyms ”computer” and ”hostname”. While the record of the ”slogans” machine in topic map is an instance of the Machine topic-type. The ”slogans” topic is the end point, meaning that there are no topics of type ”slogans”.
A topic acts as an autonomous agent, it decides itself what associations will establish with other topics. The number of associations a topic can establish is unlimited. Associations must reflect the state or relation of the object, which a topic represents with other objects.

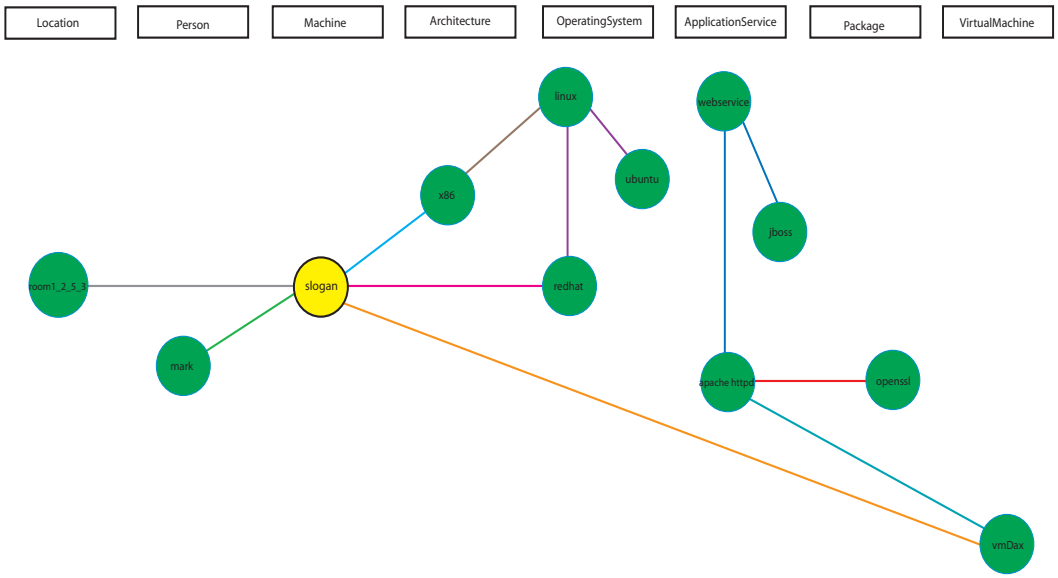

Figure 5.4: The diagram depicts associations of topic ”slogans” with its related topics.

Rectangles depict topic-types (they are also topics), while circles depict instances of a particular topic-type. Topics were disposed under a given topic-type to indicate their scope. The colored lines between topics indicate different association types that exist between topics. ”slogans” of topic-type ”Machine” is the topic in focus. It establishes an association of type *Ownership* with the topic ”Mark” of topic-type Person. Whereas, with topic ”x86” of topic-type Architecture, the ”slogans” establishes an association of type *hasMotherboard* . Topic maps are very rich with expressiveness, they can draw links beyond topic-types. The association between ”slogans” and ”Mark” goes beyond topic types. While, the association between ”linux” and ”ubuntu” is within a topic-type. Topic maps are referred to as a *subject-centric* model, since the focus of the model is the subjects.

5.3 Results

5.3.1 CMDB: optimized for updating

Optimization is carried out with regard to increasing the efficiency of a system. A system might process different types of events, which occur at different and incomparable rates. It is crucial to consider this fact when designing a model for a given system. Frequent events are most important events to consider when designing a model, because they put most of the burden on the system. The state of components of a datacenter’s environment changes often. This

means that the CMDB has to be updated often as well. Therefore, our CMDB is designed with purpose of optimizing updating.

An analogy of database optimization, but with aim of enhancing read performance is implemented in the Lightweight Directory Access Protocol (LDAP). LDAP is an authentication system that is queried to read credentials during the authentication process for a user. By contrast, adding of new users (updating) is a rare event.

Standard relational databases are optimized for searching data. Meanwhile, updating is costly, due to the fact that the updating can require adding data on more than one table. In addition it searches for matching data values between related tables.

In entity relation model, one models services running on machines using three entity relations, precisely machine entity, service entity and promisedService entity. When we register new machines and their services they promise to deliver, the insertion process has to be proceeded according to the principle that inter-table has to be populated last. This implies dependency. In addition, it introduces the database performance issue. The insertion or updating mechanism reads records sequentially in the Machine and Service tables in order to fulfill primary-foreign key matching criteria.

The update for promisedService table is of order:

$$O(M + M + N + logQ) = O(M + N + logQ) \tag{5.1}$$

where (M + M) is for checking machines twice (promiser and promisee).
N accounts for checking Services once. logQ accounts for checking for duplicates in promisedService itself, and it logQ becuase the check is indexed.

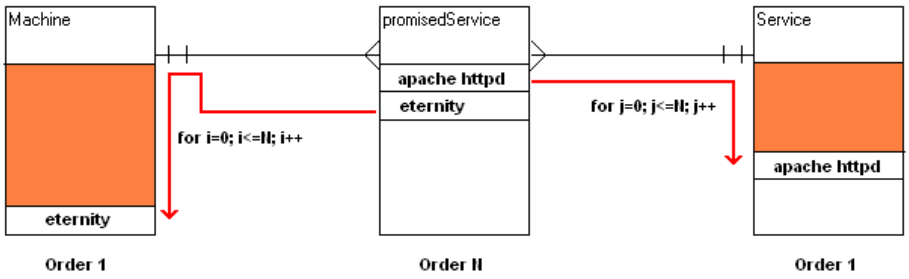

Figure 5.5: Schematic representation of updating promisedService table (standard relational database)

The orange area indicates traversing of records within the tables for purpose of matching the right values, before adding or updating occurs in the promisedService table.

Updating of some part of a CMDB (e.g. monitoring) is a frequent event. In order to boost its performance we excluded inter-tables which connected main tables. In our CMDB model, the main tables were labeled as database boxes. They were linked together through an unique value (ref. figure 6.4). Non-application of primary-foreign key mechanism in our CMDB model impacts searching performance. As an important mechanism of standard relational databases, it has been optimized for extracting information from databases. It does not introduce huge problems (querying of CMDB is not frequent) compared to the significance of optimizing dynamic updating of CMDB.

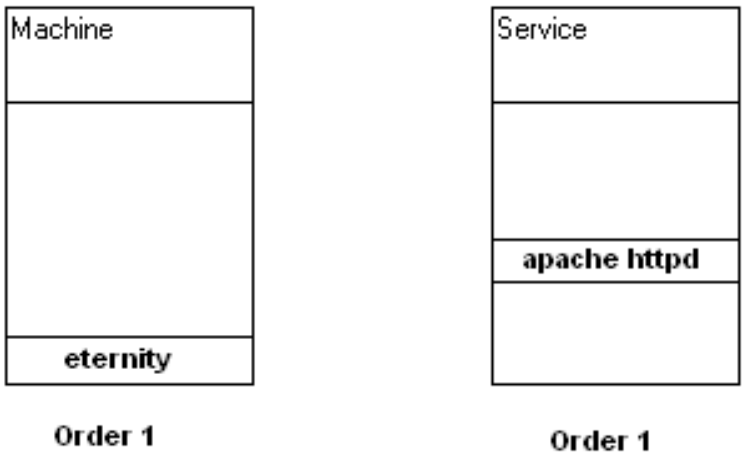

Figure 5.6: Schematic representation of updating database boxes (our CMDB model).

As we can see from the Figure 5.6 the order of adding or updating data in our database boxes is O(1).

From the standard relational model, we observe that in order to find hostname of machines running Domain Name Service (DNS), it is required to perform two join operations (between Machine and promisedService; promisedService and Service). The searching process includes examining records of all three tables.

The searching order then would be:

$$O(M * N + N * S) \tag{5.2}$$

.

M accounts for machines (Machine table), N accounts for machines promising services (promisedService table), and S accounts for services (Service table). M*N is because of the searching between the Machine and promisedService tables. While N*S accounts for searching between promisedService and Service tables. Then, we add the performance of these operations.
As in updating process, the Service table (S) does not impact searching performance significantly. It is due to the small number of records the Service table contains.

Meanwhile, in our CMDB model, It is required only one join operation (between Machine and Service). They are connected through the unique value labeled motherboardSerialNo. In the Machine box it used as a primary key, whereas in Service box it is just a common attribute, added for purpose of connecting database boxes. The searching process involves reading of both database boxes, which means that in worse case searching order would be

$$O(M * N) \tag{5.3}$$

where M accounts for machines (Machine database box), and N accounts for services (Service database box).

To enhance the searching performance, we index the common identification attribute residing in Service database box. This makes querying our CMDB approximately as efficient as the standard relational databases.

e.g. in order to find all machines running DNS we write the SQL query:

```
select machine.hostname
from machine, service
where machine.motherboardSerialNo = service.motherboardSerialNo
and service.serviceType = 'DNS'
```

As we can see from the SQL query the motherboadSerialNo attributes are used to join the database boxes.

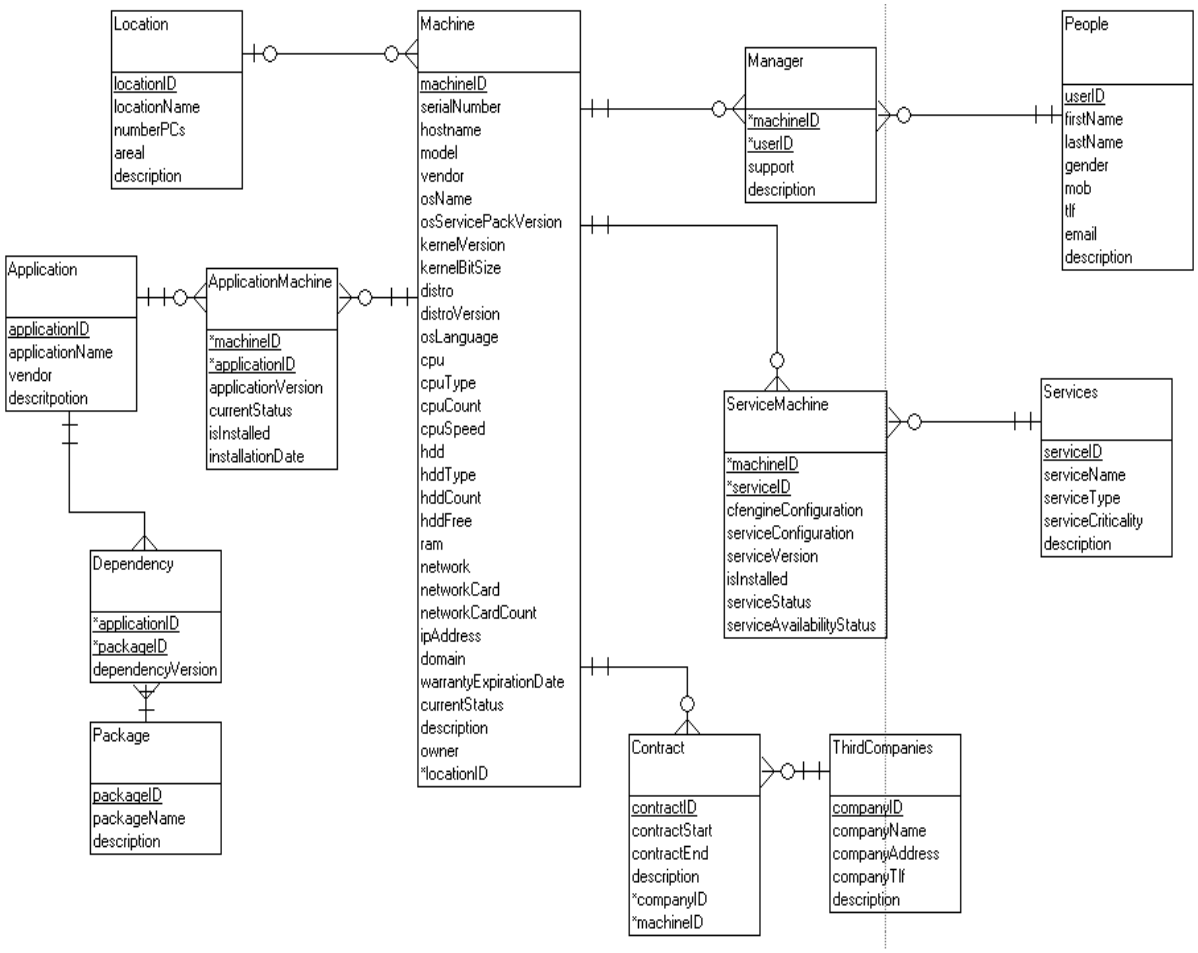

Figure 5.7: The Single CMDB draft designed based on strict application of relational model principles.

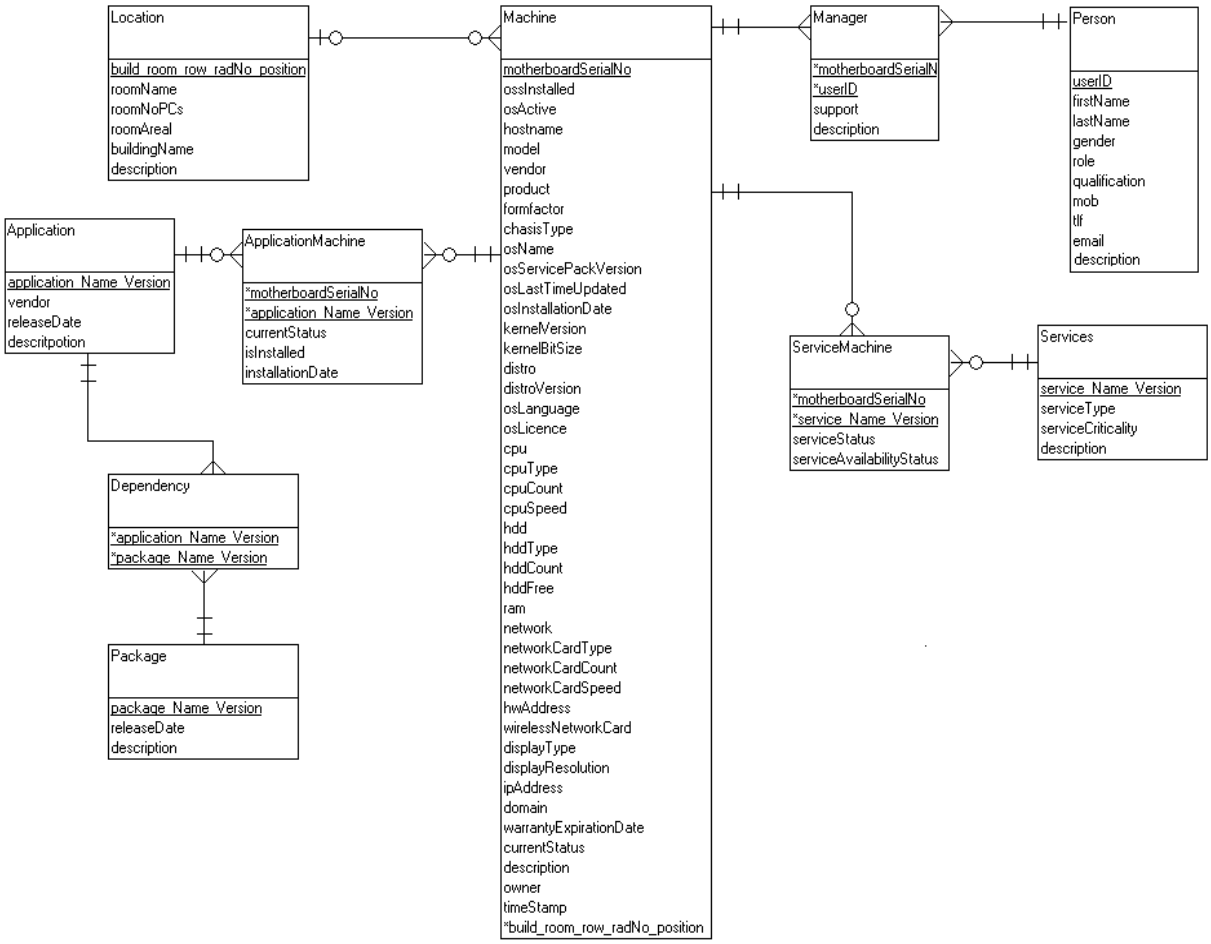

Figure 5.8: Exclusion of centralizing electronic documents. This CMDB model does not contain entities for storing documents. They are accessed via topic map occurrences.

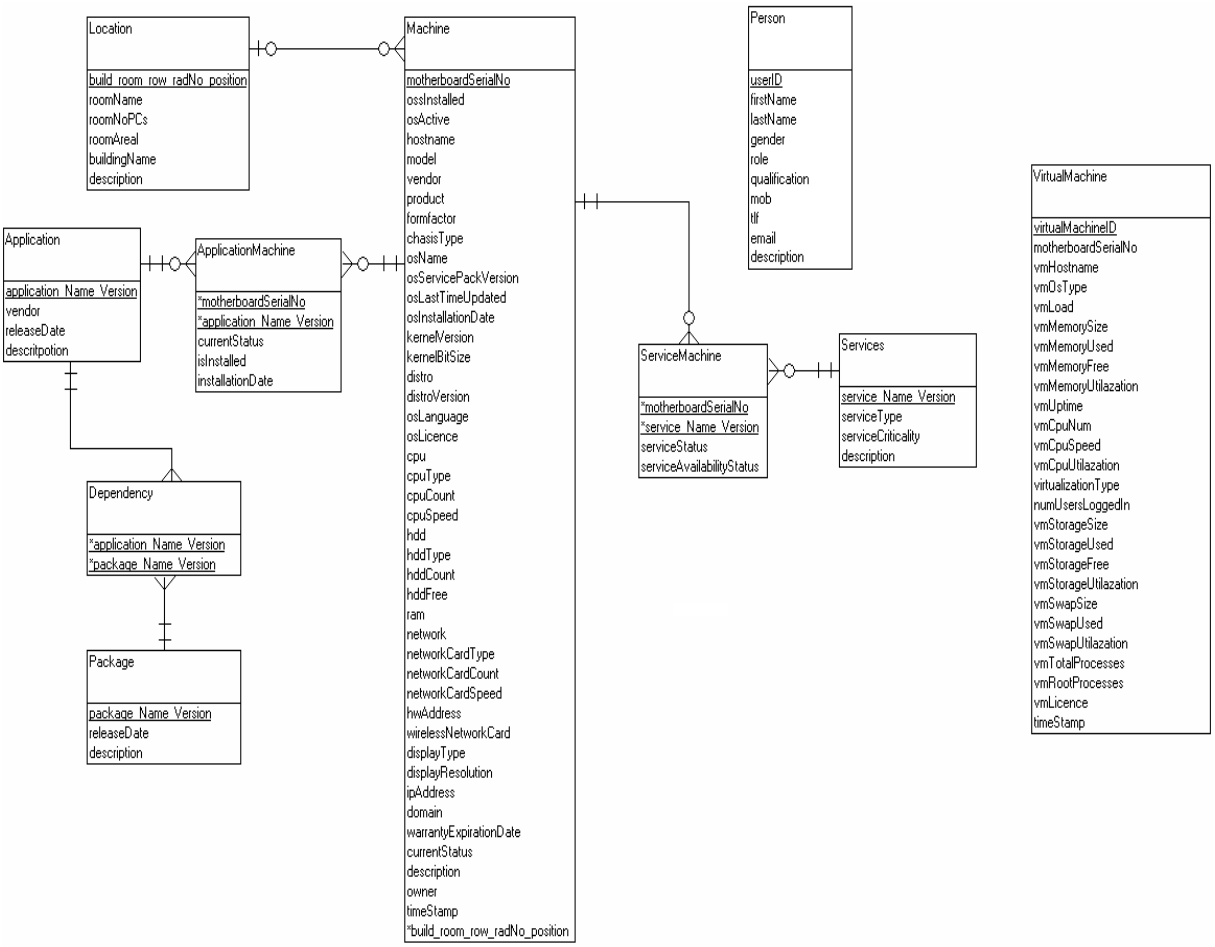

Figure 5.9: The three databases CMDB model modeled based on the rate at which data changes. It is consisted of three databases: the standard relational database (slowly changing data), the People database (manually-entered data), the VirtualMachine database (monitoring data)

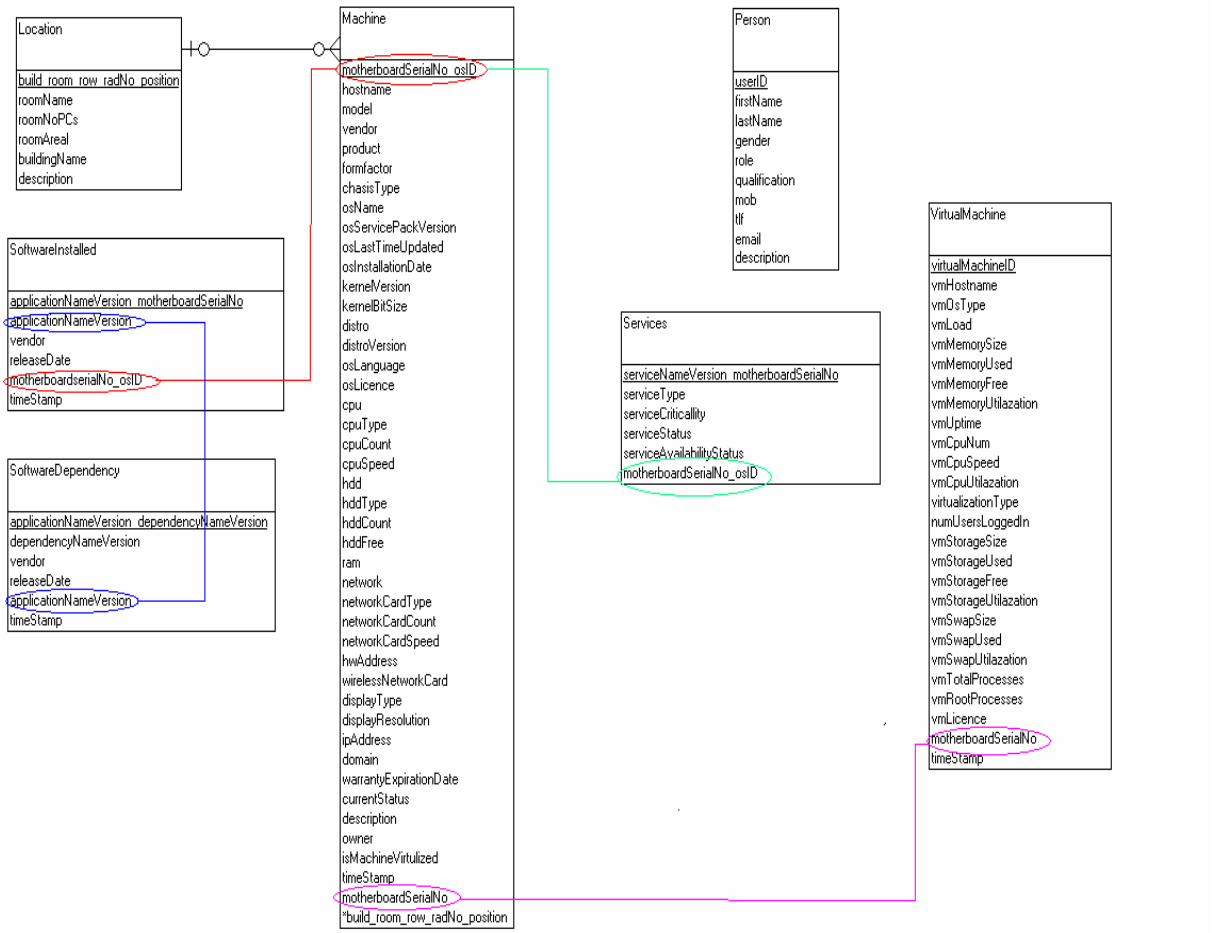

Figure 5.10: The introduction of ”box concept” in our CMDB model. Database boxes are connected to each other through logical unique values (common attributes). It is indicated by the colored lines

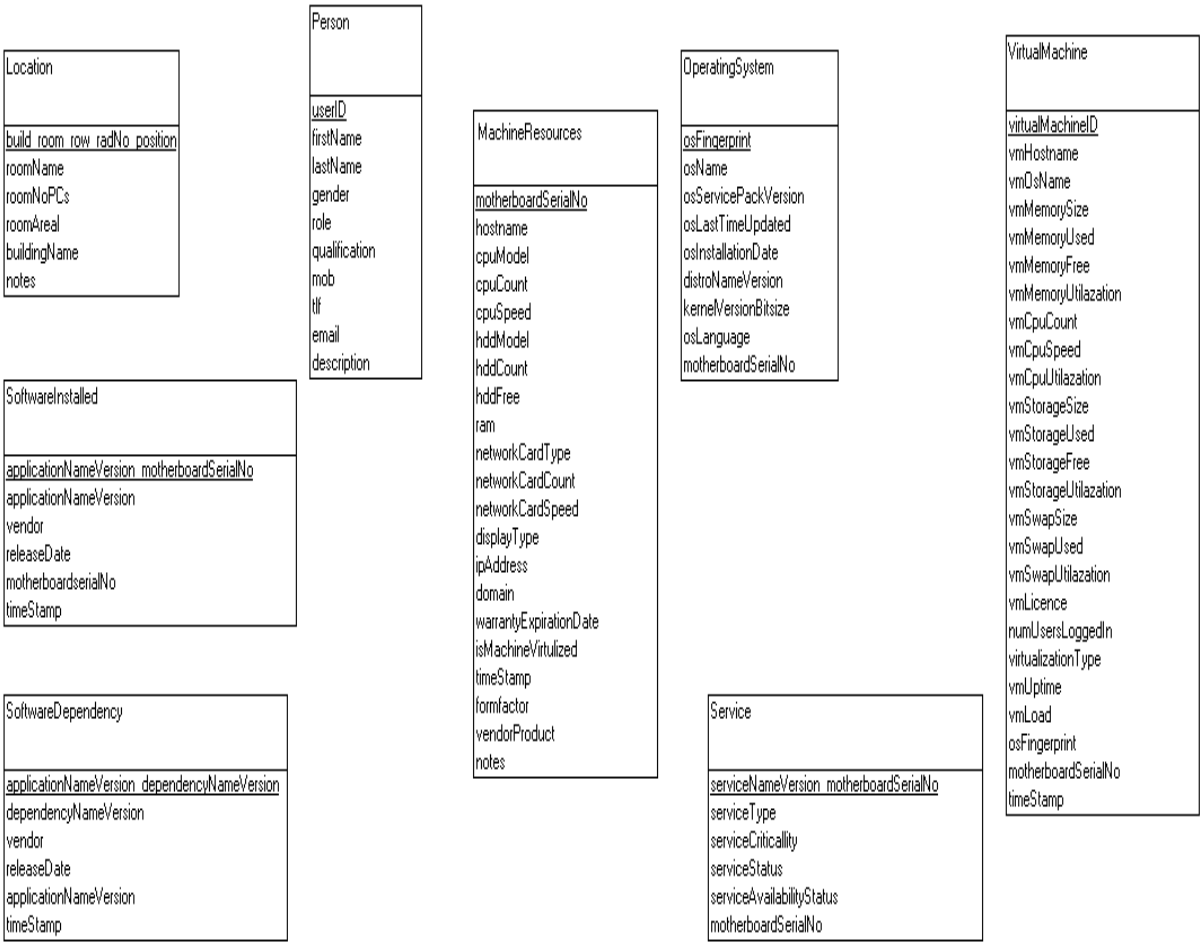

Figure 5.11: Splitting of Machine database box into MachineResources and OperatingSystem database boxes

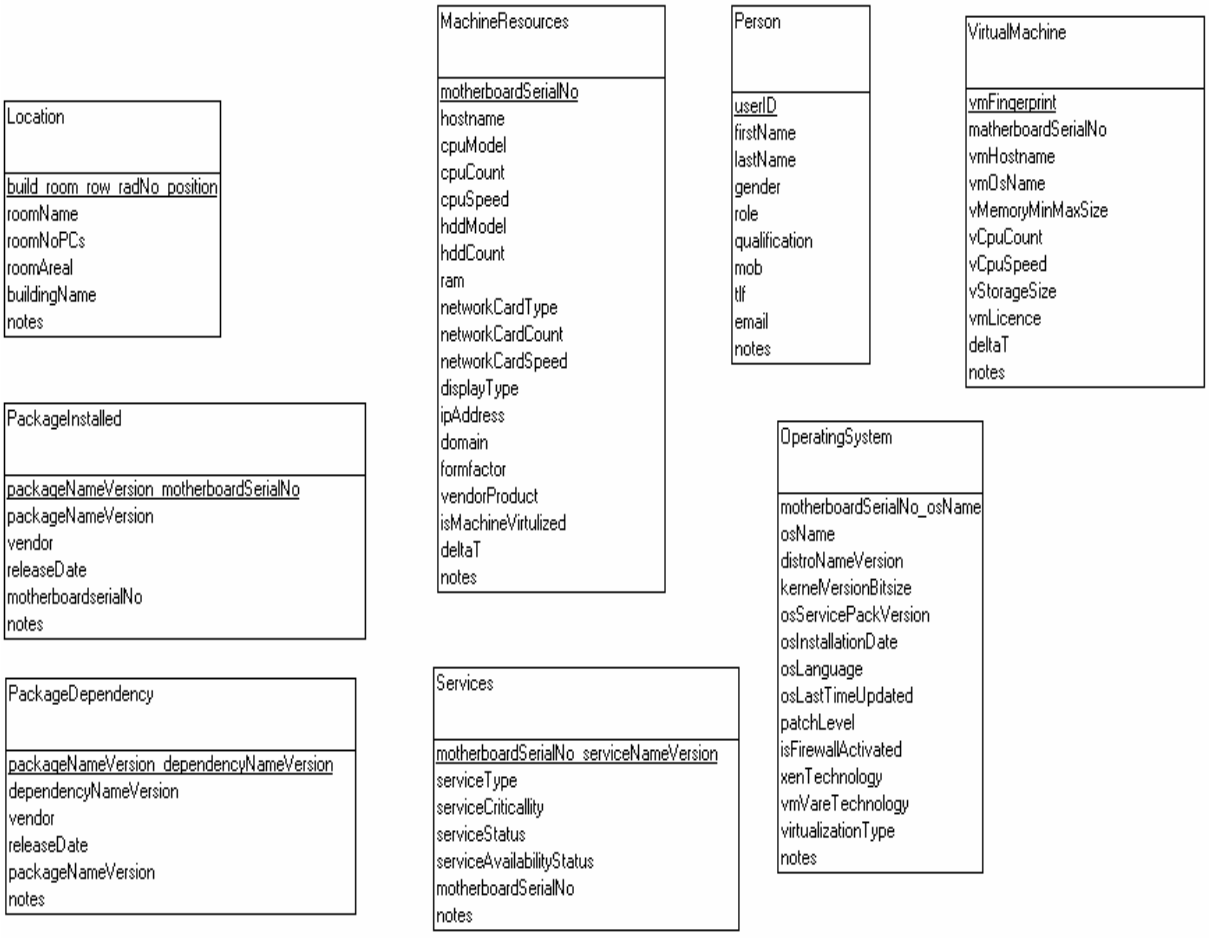

Figure 5.12: Removal of monitoring data from the CMDB

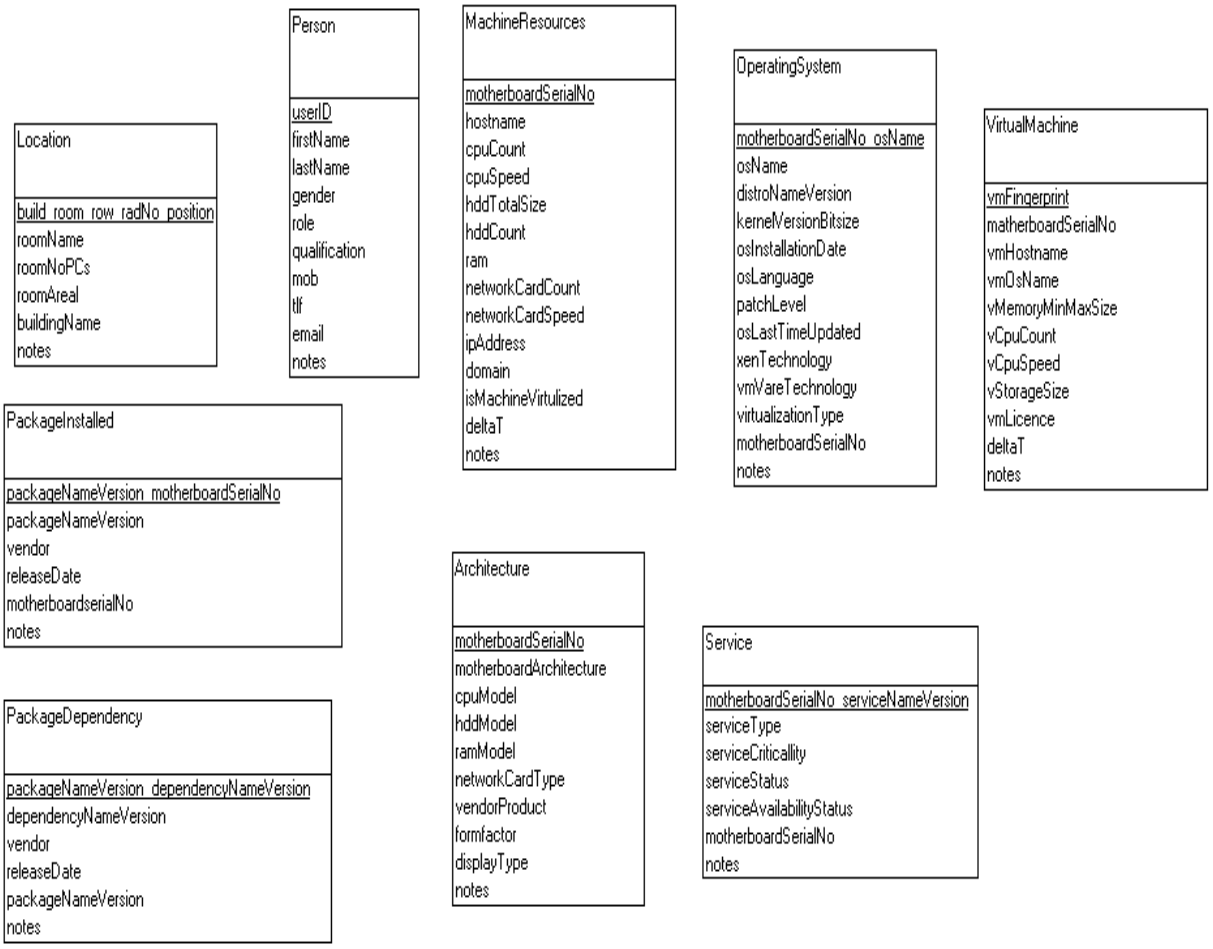

Figure 5.13: The introduction of change *concept* in our CMDB model.

Chapter 6

Test case, Evaluation, and Discussion

This test scenario was created in order to demonstrate the behaviors of our CMDB in real implementation. In the discussion section we have discussed the characteristic features of our CMDB model. An interesting point is made by comparing it with the star data warehouse model.

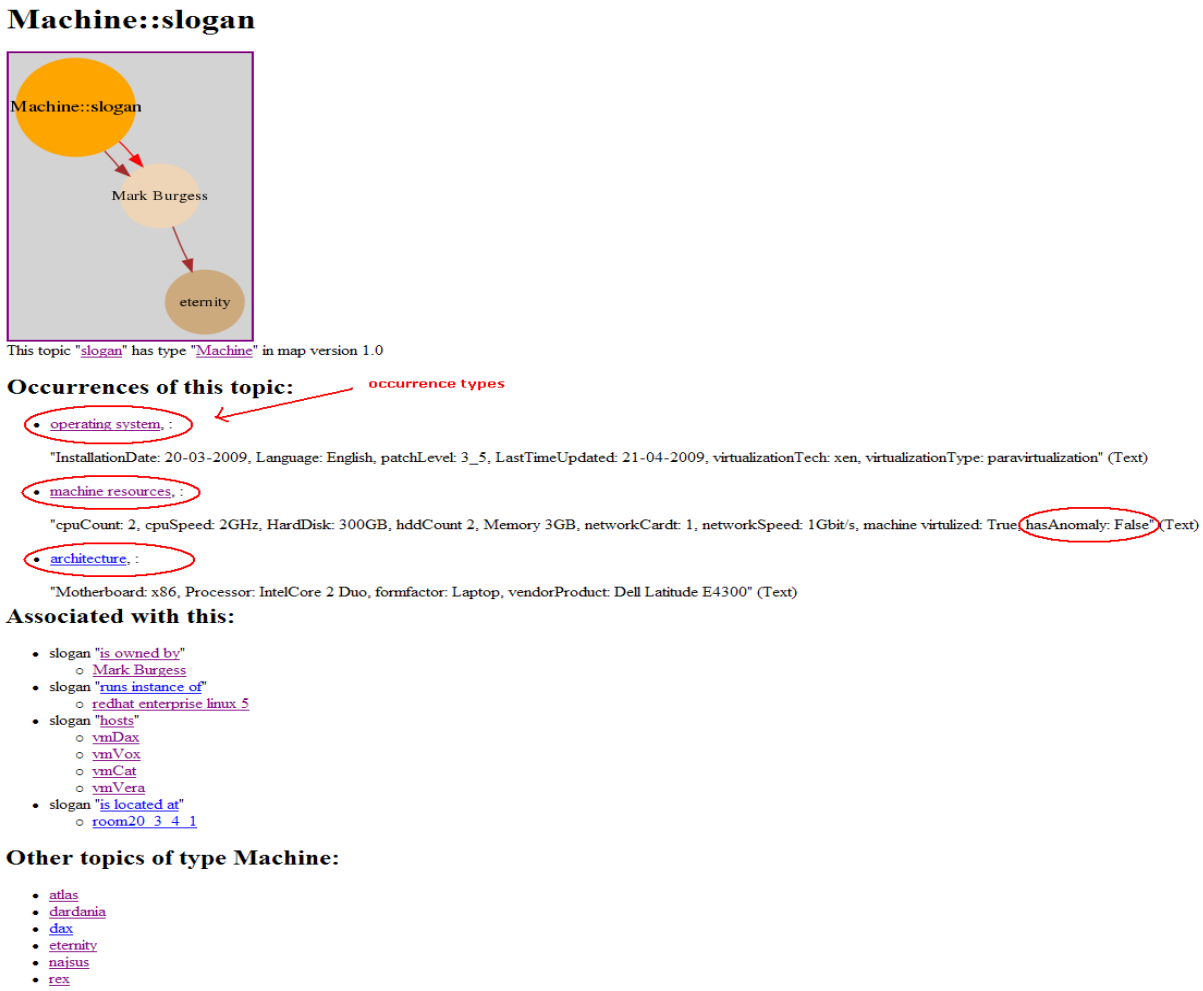

Figure 6.1: The ”slogans” topic, its occurrences and associations. From the occurrences, we specially stress the anomaly detection, since this feature is not supported by existing CMDB solutions.

The Figure 6.1 displays all information of the ”slogans” topic. The types of information include occurrences, the associations and other topics of type Machine. Occurrences are typed; this allows us to understand what the data is about. This means that we can just refer to the occurrence type to see if we are interested in that type of information. From the occurrence *operating_system* type, we observe that the ”slogans” machine has been virtualized in paravirtualization mode. While in the association section (in addition to other associations) were listed the virtual machines that the ”slogans” machine hosts. If we visit one of these topics we will get the semantic and descriptive information (occurrences) on that topic. In this scenario we visited the ”vmDax” topic, and we were moved to a different point in the topic map.

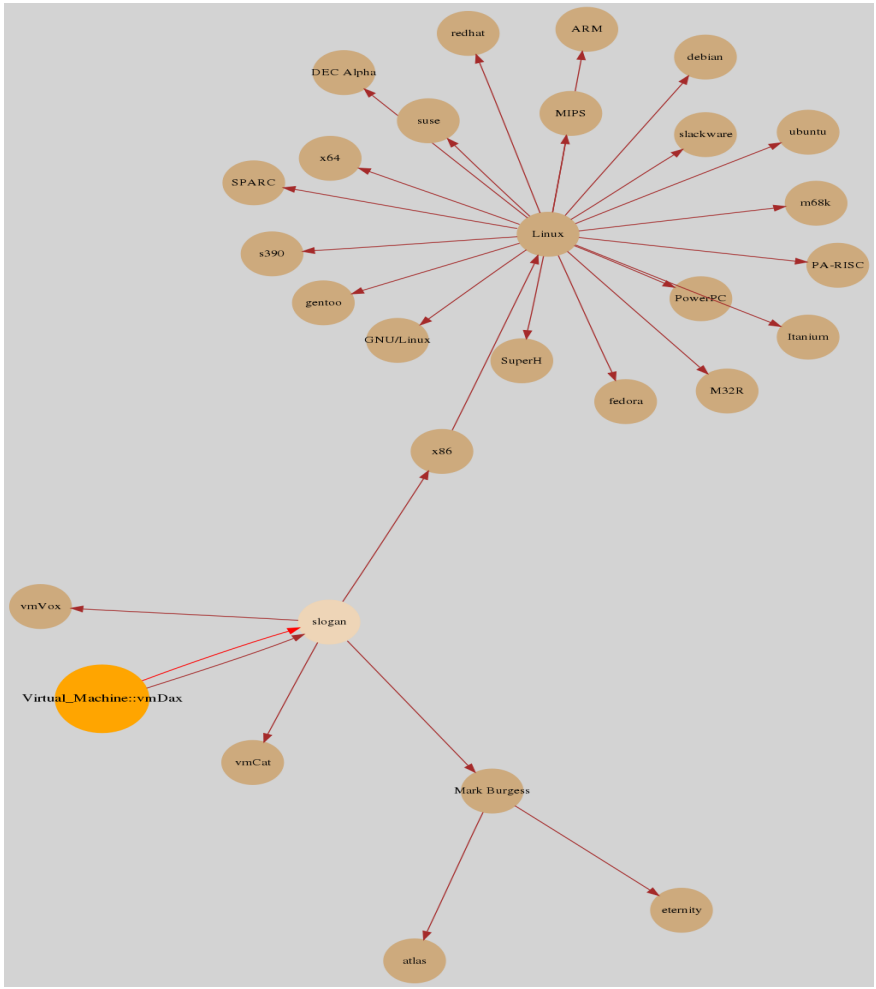

Figure 6.2: vmDax

The red arrow indicates the directed topic(s) of the topic in question. From the Figure 6.2 we can observe that the ”slogans” machine in addition to hosting ”vmDax”, it also hosts ”vmCat” and ”vmVox”. The semantic association allows us to know more about the topic. Here among other we get information about the architecture and owner of the machine.

Figure 6.3: vmDax2

From the Figure 6.2 we can view that the ”vmDax” virtual machine has two types of occurrences, namely the virtual resources (residing in the CMDB), and monitoring data residing in the virtual machine. By selecting the link of monitoring data, we retrieve this information from the remote vmDax, see Figure 6.4. This is one of the unique features of our CMDB solution. In the literature research we found no existing CMDB solutions to provide such a capability.

Keeping the monitoring data in the local agents was quite important approach implemented in our CMDB model. Since, the centralization of the monitoring data of a datacenter’s environment is not a clever approach, because the amount of information is enormous.

The Figure 6.2 captures also an important aspect of our CMDB approach. In topic map we stored information on operating system such as the hardware architecture supported, distributions or versions they have. Refer Figure 6.5 to view relations of ”windows” with its versions. In this example, the direct relations of the ”Linux” topic indicate the hardware architectures that instances of Linux operating system can run on, and its distributions. This was the right approach to pursue; here we list the arguments for this decision:

- *Our CMDB model became simpler*
- *Our CMDB will be re-created automatically, whereas they were manually en-*

tered data

- *they were static data and small in quantity*

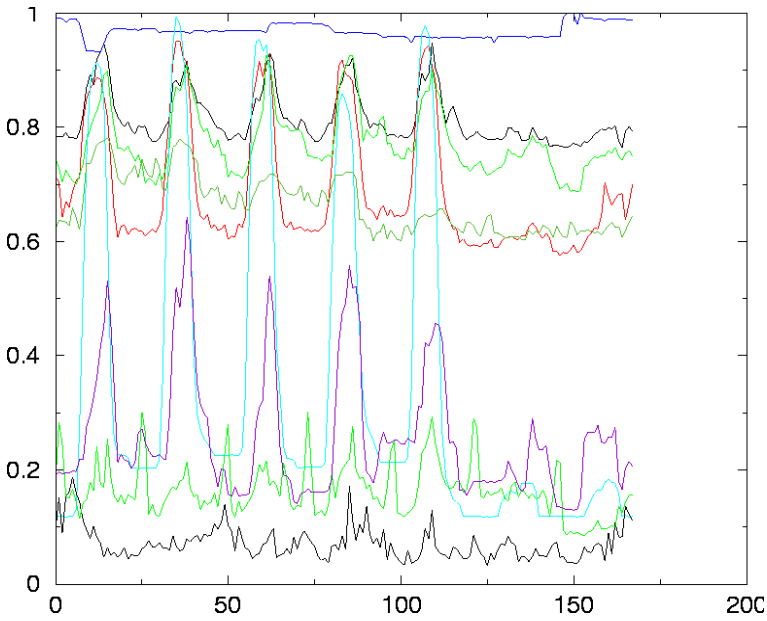

Figure 6.4: A track of multiple scaled system variables as the number of users and processes, free disk space, www connections, ect., over the course of a week [18].

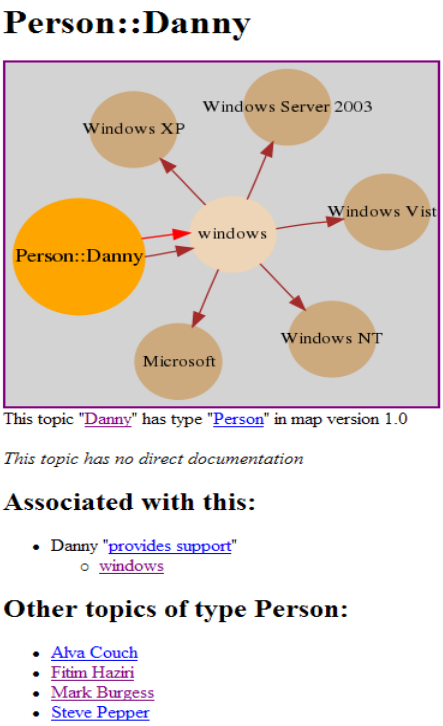

Figure 6.5: OS

Our topic map CMDB solution allows one to interpret the data according to their needs. In the Figure 6.5 is shown that ”Danny” *provides support for* ”windows”. While in the Figure 6.2 is shown that the ”slogans” machine *is owned by* ”Mark”. Occurrences of these topics representing different subjects reside in the CMDB schema, but using the promise approach one can build a semantic topic map layer in a convenient and flexible manner.

To express the semantic relationship of a topic, and point to its occurrences one needs to add only a single line in the policy configuration file. The following Cfengine’s topic map promise adds the topic ”Danny” into the topic map layer.

```
Person::

"Danny" association => a("provides support for", "windows", "supported by");
```

6.1 Discussion

Enterprises possess different types of information, starting from configuration information to contracts, documentations, manuals, reports, monitoring data etc. This diversity of information and the manner it is obtained has influenced

us in drawing the principles for modeling our CMDB model. The compilation of principles for modeling our CMDB has led us to design a model with unique characteristics.

Our CMDB model is designed based on ”intersecting” of the entity relation model, and topic maps model. They are two different design information technologies, which have their own strengths and weaknesses in certain areas. It was important to exhibit the features of these models so that we could reflect their strengths in our CMDB model, and avoid pitfalls.

6.1.1 CMDB drafts

The single CMDB model was completely built within principles of entity-relation model. It became complicated and provided constrained capabilities. The single CMDB model was built up of twelve different entities and supported only six types of services. The process of collecting data was not efficient. This approach seemed inefficient in extending further capabilities to CMDB. This type of CMDB model is offered by vendors today.

Exclusion of electronic documents from being modeled in entity relation model, was the first step towards simplifying of our CMDB model. But the impact was minimal, and consequently the CMDB draft remained complex.

In the three databases CMDB model, we tried to introduce a new approach in designing our CMDB model. We separated data into different databases based on the rate at which data change. This made the three databases CMDB a bit more efficient in collecting data. One major problem with this CMDB draft is that the configuration information (slowly changing data) database inherited all drawbacks of its precursor CMDB drafts.

The introduction of box concept in our CMDB model addressed important issues such as dependency (inter-tables), breadth and depth complexity, and optimized updating process of our CMDB. The problem with this CMDB draft was that the Machine database box contained unclassified information, apart from introducing depth complexity.

Splitting of Machine database box into MachineResources and OperatingSystem database boxes was conducted for purpose of classifying information into different database boxes. This would allow us to communicate data to database boxes that the rate they change. The side effect of this step was that the depth complexity of Machine database box was reduced. A major issue with this CMDB model was inclusion of monitoring data. It is very expensive to centralize monitoring data, or impossible in large scale networks.

Monitoring data are excluded from being stored in the CMDB. They are generated on demand and stored in the local machines. The generated reports

are accessed through topic map occurrences. This approach removed bottlenecks in our CMDB. In this draft we added the *notes* field in order to make our CMDB model extensible. Occurrences of *notes* field are linked back to topic map forming a fully integrated index. This CMDB draft supports anomaly detection capability, which is indicated by the *deltaT* attribute residing on the MachineResources and VirtualMachine database boxes. In OperatingSystem database box we added the capability of storing the firewall status. The flaw in this CMDB draft was that the MachineResources database box contained some unclassified information, namely, static and slow changing data.

Features of the ultimate version are discussed in section 6.1.3.

6.1.2 Non-hierarchical

Topic map model is rich in capabilities. It provides a framework for designing the information in hierarchical and non-hierarchical fashions. Using the flexibility of topic map, the information in our CMDB model is organized in a ”flat” fashion, meaning that there is no hierarchy. Instead of constructing hierarchical structures, related concepts are grouped together in a semantic manner.

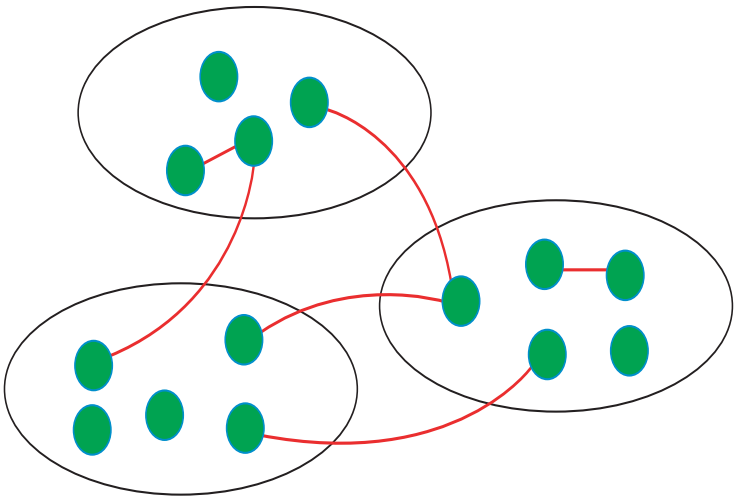

Figure 6.6: Topic map, CMDB: grouping of topics

This feature is not found in traditional CMDB models. They organize information in hierarchical structure. This approach is characterized of splitting objects apart into different branches. This introduces the problem of adding additional capabilities later on, since they have separated properties and behaviors that they might need. To include the new capabilities they have to redesign the model.

IBM tries to avoid this problem for its CMDB solution by retaining different types of constructs into different containers. Attributes are stored separately

from classes with the purpose of being able to reuse in any number of class definitions [5]. In addition, accessing of objects in hierarchical structure is costly. The journey of accessing the information has to go through neighbors.

6.1.3 Characteristics of our CMDB model

Our ultimate logical database model incarnates the best practices of process development . Here we list the unique features that our CMDB model possesses:

- **Simple to understand:**

  We have compiled the principles for our CMDB model followed by a metaphor, which makes it very easy to understand. See section 5.1.

- **Classification of configuration information:**

  This feature is rather important, since it enables cfengine agent to communicate the configuration information in an efficient manner. So, the static configuration information is communicated only once, because they do not change. Therefore, can say that there is a correlation between the information theory and database boxes in the sense that information that does not not change makes no sense to retransmit. There is no new information in the transmitted message.

- **Database boxes reflect human limitations with regard to processing of information:**

  Here we applied the researching results of British anthropologist R. Dunbar who gave the threshold of human in maintaining reliable relationships with individuals. This was rather important since Dunbar exhibits limitation of human in processing information. The fundamental principle that many scientists highlight is that in the CMDB should not be stored information that is unmanageable. Precisely, this was taking into account during the design of our CMDB model.

- **Extensible:**

  We can attach a new database box into the model, and it will not affect the rest of CMDB. This is achieved due to each of database boxes is independent. Our CMDB model is also extensible through the *notes* field. The latter approach was quite important since it removed the need of redesigning the model.

- **Easy to update:**

  Considering the fact that in the large scale and dynamic networks there is a lot of configuration information that need to keep track of CMDB. This really means that we have to update the CMDB frequently. Our approach was unique in addressing this issue, by optimizing our CMDB model for updating process No one else has optimized CMDBs for purpose of boosting the updating process. In addition, the configuration

information will be communicated to the CMDB at the rate they change. (Timescale is a feature of our CMDB.)

- **Scalable:**

  database boxes are scalable, inside these can be stored large amount of information.

In literature we have found that one of the reasons that CMDB implementations fail is due to collecting of detailed information [19]. They cannot manage this information, and consequently do not provide a ROI (Return Of Investment). It is wrong to store information, which is unmanageable. It will only make the CMDB more complex. Meanwhile, our CMDB is characterized of storing information at the granularity which human are able to process effectively. Another issue that we exhibit from literature section is that existing CMDB solutions do not recognize the *timescale* feature. While cfengine learning approach does support this feature. It communicates data in real time and at proportion with the rate data changes.

Database boxes of our CMDB consist virtually of two dimensions. The configuration information of first rank resides in appropriate attributes, while the *notes* filed contains other configuration information that cfengine agent discovers. The introduction of the *notes* field was important approach, since it eliminates having empty fields in the database boxes. There is mapping between the configuration item and the attribute field. In cases where the configuration item does not exist in all instances of a give entity type, then it is moved to the *notes* field. But the real power of the *notes* fields lies in the fact that it makes our CMDB model extensible without having to redesign the model. Occurrences of the *notes* filed will be designed in the topic map structure and linked to the topic map layer.

6.1.4 How does our CMDB model fit with other models ?

It was an interesting approach to view our CMDB model against other database models, relate with them and observe the similarities and differences that exist among them. Considering the fact that our CMDB solution will function as a data warehouse it was natural to compare it with data warehouse models.

The star model is the most common model for constructing of the data warehouses. An example star model for property sales is shown in Figure 6.7. The star model is consisted of a central table called the *fact* table, and a set of small table called *dimension* tables. The *fact* table can be extremely large, since it holds the "facts" that is going to be analyzed. While the *dimension* tables contain descriptive information used as the constraints in data warehouse queries.

In our CMDB model we can install the *MachineResources* database box at the center of the model connecting it by other database boxes through an unique

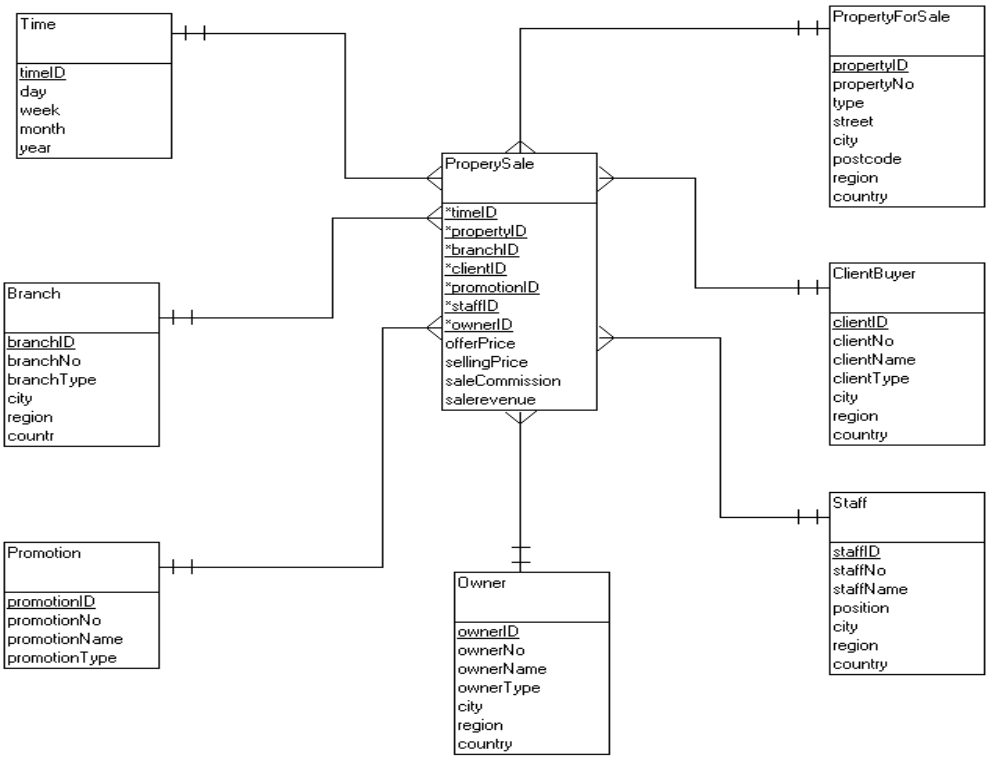

Figure 6.7: Star model

logical value forming a star-like model, see Figure 6.8. Unlike to the data warehouse star model, the *facts* in our CMDB model are scattered in other database boxes as well.

The other similarity between these two models is that there is always one join operation between the central agent, and its satellite agents. An important aspect of these two models is that they are not complex in breadth and depth. As we can see from the Figure 6.7 the fact table (the most important one) contains only the values which will draw the business picture of the enterprise. The same approach was pursued in designing of our CMDB model by including only the aspects that are strongly linked to increasing efficiency in the operation of the datacenters, so that they become more competitive in the market.

In the standard entity relation models, and traditional CMDB models we are accustomed to see tables and classes containing large number of elements deviating from achieving the intended goals. As an example, IBM's CMDB version stores detailed configuration information for components. This type of information cannot be utilized for enhancing the business success, on the contrary it introduces cost, since the configuration information has to be maintained.

A distinctive characteristic of our CMDB model in relation to the star data warehouse model is that the edge database boxes can be connected to each

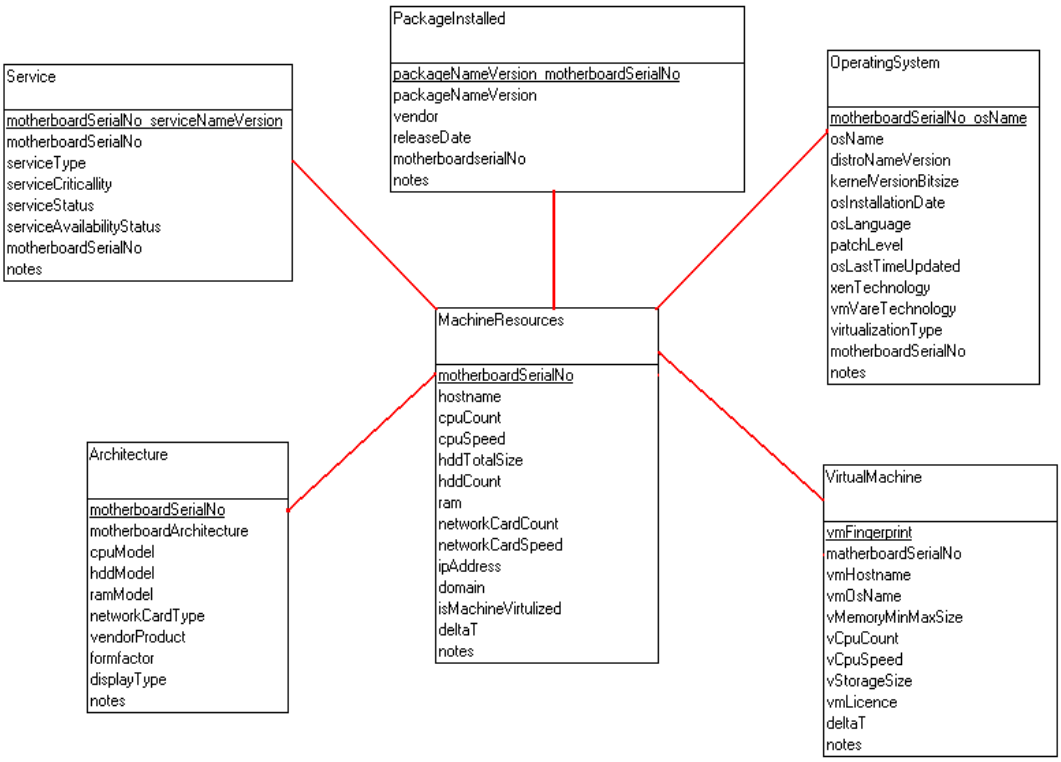

Figure 6.8: Our CMDB

other. This is achievable because we used an univaral unique key to link them. This is a flexibility that our CMDB model possesses. As an example we can extract information from the database boxes such as the Service and Package-Installed for purpose of finding packages required to set up apache httpd service.

Chapter 7

Conclusions

The aim of this master thesis was to design a Semantic CMDB model for Cfengine 3. Traditional CMDB solutions are built based upon hierarchical data models. These models are complex in structure, and the implementation process invokes serious efforts. The hierarchical modeling approach follows a strict procedure. Behaviors will be separated into different branches. An error in modeling or an incompatibility in adding new elements requires to redesign the model.
One should not constrain his view when designing models. Viewing of the entities' aspects from different perspectives allowed us to understand more about their phenomenons. This facilitated us to adjust our approach in relation to the nature of the phenomena.

Our research work has shown that the models do not need to be complex in order to provide rich capabilities. We succeed to design a CMDB model which is characterized of being simple and simultaneously provides numerous capabilities. The test case showed that the pursued approach is promising in the real implementation.

Our approach towards CMDB was radical compared to existing CMDB solutions. Given the limited time we had for this revolutionary CMDB design, we know that the model will change. This CMDB model will be improved in the future as we understand more.

7.1 Future work

One aspect of our CMDB model that needs to be addressed is structuring of the *notes* fields. The function of *notes* fields is to build a fully integrated topic map index. This would be achieved through linking the *notes'* occurrences to the knowledge layer. The work that remains is to determine a pattern that will distinguishes topics along with their occurrences and associations.

The final piece is the implementation of the designed CMDB model in Cfengine 3.

Bibliography

[1] Martin Sailer Michael Brenner, Markus Garschhammer and Thomas Schaaf. Cmdb - yet another mib? on reusing management model concepts in itil configuration management, 2006.

[2] N. Karnik A. Kunmar. Moving from data modeling to process modeling in cim. Technical report, ibm.com, 2005.

[3] V. Tosic and S. Dordevic-Kajan. The common information model (cim) standard - an analysis of features and open issues. Technical report, 1999.

[4] HP. An insider's view to the hp universal cmdb. Technical report, 2008. 4AA-1-5976ENW.

[5] R. Baker N. Ayachitula L. Shwartz M. Surendra C. Corley M. Benantar H. Madduri, S. S. B. Shi and S. Patel. A configuration management database architecture in support of ibm service management, 2007.

[6] ibm. Deployment guide services: Ibm tivoli application dependency discovery manager v7.1, 2008.

[7] R. Hill w. Zhou, D. Sornette and R.I.M. Dunbar. Discrete hierarchical organization of social group sizes. In *Proceedings of the royal society*, page 439, 2004.

[8] T. R. Gruber. *A translation approach to portable ontologies*, chapter Knowledge Acquisition, pages 199–220. 1993.

[9] J. Strassner. *Handbook of Network and System Administration*, chapter chapter, Knowledge Engineering Using Ontologies. Elsevier Handbook, 2007.

[10] Steave Pepper. *Encyclopedia of Library and Information Sciences*, chapter chapter, Topic Maps. CRC Press, 2009.

[11] Steve Pepper. The tao of topic maps, 2000.

[12] Mark Burgess. Knowledge management and promises, 2009.

[13] Peter P. Chen. Entity-relationship modeling: Historical events, future trends, and lessons learned.

[14] Th. Connolly and C. Begg, editors. *Dataabse Systems*, page 1209. University of Paisley, 2005. ISBN 82-579-4155-7.

[15] Mark Burgess. An approach to understanding policy based on autonomy and voluntary coorperation, 2005.

[16] Rolf Guescini Dino Karabeg and Tommy W. Nordeng. Flexible and exploratory learning by polyscopic topic maps, 2005.

[17] Omg, unified modeling language, version 1.5, 2003.

[18] Sigmund Straumsnes Hrek Haugeud. Simulation of user-driven computer behavior, 2001.

[19] Buff Scott. Practical advice on implementing your configuraton management database, 2008.

Appendix A

A.1 Summary of boxes' capabilities

So our model consists of a number of boxes containing information. Let us describe the contents of these boxes systematically.

A.1.1 The machine box

From a complex container with 44 different elements at the beginning, we have turned it to a small and powerful box with only 12 elements. Here we list the main capabilities of machine resources box:

- *Machine's resources*
- *Information whether machine is virtualized*
- *The network machine is connected to*

A.1.2 The operating system box

The *operating system box* provides the most useful capabilities. Here we summarize some of them.

- *Description of operating system*
- *Patch level and security updates*
- *Virtualization type implemented*
- *Virtualization technology running*
- *Anomaly detection*

A.1.3 The virtual machine box

The *virtual machine box* serves as an inventory of virtual machines on a datacenter. Here we list the main views that are available for virtual machine box.

- Where virtual machines are deployed
- Resources that they have been given

- To which virtualization technology they belong
- Details of anomalies
- License compliance

A.1.4 The architecture box

The architecture box would be used mainly for purposes of:

- *Ordering machine's components from the vendors*
- *Determining the proper components when upgrading machines or running special programs*

A.1.5 The service box

The service box provides this information.

- *Service types the machine promises to deliver (web, dns, email)*
- *The technology that is installed to deliver the service (apache, jboss)*
- *The criticality of service (high, middle, low)*
- *The status of service (installed, not installed)*
- *The availability status of service (active, non-active)*

A.1.6 The PackageInstalled box

- *Packages installed on machines, including its vendor and version*

A.1.7 The PackageDependency box

- *Dependencies of installed packages, including their vendor and versions*

Appendix B

B.1 Cfengine configuration code for the test case

```
#############################################################################
#
# Knowledge base / ontology
#
#############################################################################

body common control
   {
   bundlesequence  => {
                       "tm" ,
                       "manual_entries"
       };

   version => "1.1";
   }

#############################################################################

body knowledge control

{
graph_output => "true"; # override -g
graph_directory => "/var/www/graphs";

build_directory => "/var/www/";
#id_prefix => "cfengine";


sql_server => "localhost";
sql_database => "cf_topic_map";
sql_owner => "root";
sql_passwd => "kos17"; # No passwd
sql_type => "mysql";
```

```
query_output => "html";
query_engine => "index.php";

style_sheet => "cf_enterprise.css";

html_banner => "<div id=\"top\">      <div id=\"search\">
        <form method=\"post\" action=\"$(query_engine)\">
          <p><input class=\"searchfield\" type=\"text\" name=\"regex\" /></p>
        </form>
      </div>
<a href=\"$(query_engine)\"><img src=\"KnowledgeBanner.png\" border=0></a></div>";

html_footer => "<div id=\"footer\">Copyright &copy; Cfengine AS</div>";

}
```

```
#############################################################################
# The Map
#############################################################################

bundle knowledge tm

{
vars:

  #
  # Association bank
  #

  "own[f]" string => "owns";
  "own[b]" string => "is owned by";

  "moth[f]" string => "has motherboard";
  "moth[b]" string => "is a component of";

  "plat[f]" string => "supports instance of";
  "plat[b]" string => "runs on platform";

  "same[f]" string => "is related to";
  "same[b]" string => "is related to";
```

```
  "dis[f]" string => "is a distribution of";
  "dis[b]" string => "has distribution";

  "virt[f]" string => "hosts";
  "virt[b]" string => "is hosted by";

  "prom[f]" string => "promises to deliver";
  "prom[b]" string => "is promised by";

  "sim[f]" string => "has service type";
  "sim[b]" string => "is a kind of";

  "dep[f]" string => "depends on";
  "dep[b]" string => "is required by";

  "loc[f]" string => "is located at";
  "loc[b]" string => "holds";

  "vers[f]" string => "is a version of";
  "vers[b]" string => "has version";

  "ven[f]" string => "has vendor";
  "ven[b]" string => "released";

  "sup[f]" string => "provides support";
  "sup[b]" string => "support is provided by";

  "run[f]" string => "runs instance of";
  "run[b]" string => "runs on";


#
# Some related topics form a promise "group by association"
#

 "linux_distros" slist => { "suse", "ubuntu", "fedora", "redhat", "debian",
 "slackware", "gentoo" };

 "linux_platforms" slist => { "x86", "MIPS", "x64", "SPARC", "DEC Alpha", "Itanium",
 "PowerPC", "ARM", "m68k", "PA-RISC", "s390", "SuperH", "M32R" };


"solaris_2_platforms" slist => { "sun4c" };

"solaris_10_platforms" slist => { "SPARC-32", "SPARC-32", "x86-32","x86-64" };
```

```
"slogan_vms" slist => { "vmDax", "vmVox", "vmCat", vmVera };


"apache_webservers" slist => { "vmDax", "dardania", rex, vmVox };

"webservices" slist => { "apache_httpd", "jboss", "tomcat" };

"app_Service_Dep_openssl" slist =>  { "apache_httpd", "cfengine", "DNS" };

"windows_versions" slist => { "Windows XP", "Windows Vista",
"Windows Server 2003", "Windows NT" };

"redhat_enterprise_linux_5_on_machines" slist => { "slogan", "dardania", dax };


topics:


#############################################################################
# untyped topics are foundation
#############################################################################

 # Put these here as short cuts from the front page

 Start::

   "Categories";

 Categories::

   "Machine";
   "Person";
   "Architecture";
   "Application_Service";
   "Operating_System";
   "Virtual_Machine";
   "Package";
   "Location";


#############################################################################
# typed topics
#############################################################################


Vendor::
     "Sun";
```

```
     "Microsoft";
     "redhat";

Person::

    "Mark Burgess";
    "Alva Couch";
    "Fitim Haziri";
    "Steve Pepper";

    "Danny"
association => a("$(sup[f])","windows","$(sup[b])");
    "Robert"
association => a("$(sup[f])","redhat","$(sup[b])");


Machine::

   "atlas"
association => a("$(own[b])","Mark Burgess","$(own[f])");
   "dardania"
association => a("$(own[b])","Fitim Haziri","$(own[f])");
   "slogan"
association => a("$(own[b])","Mark Burgess","$(own[f])");
   "eternity"
association => a("$(own[b])","Mark Burgess","$(own[f])");
   "najsus"
association => a("$(own[b])","Alva Couch","$(own[f])");
   "rex"
association => a("$(own[b])","Steve Pepper","$(own[f])");


   "$(redhat_enterprise_linux_5_on_machines)"
association => a("$(run[f])","redhat enterprise linux 5","$(run[b])");



Operating_System::
    "redhat";
    "Linux";
    "Solaris 2.0";
    "Solaris 10";

    "GNU/Linux"
association => a("$(same[f])","Linux","$(same[b])");

    "$(linux_distros)"
```

```
association => a("$(dis[f])","Linux","$(dis[b])"),
        comment => "Linux distro $(linux_distros)";


     "windows"
association => a("$(ven[f])","Microsoft","$(ven[b])");

     "$(windows_versions)"
association => a("$(vers[f])","windows","$(vers[b])");

     "redhat enterprise linux 5"
association => a("is variant of","redhat enterprise linux","is variant of");

     "redhat enterprise linux"
association => a("is variant of","redhat linux","is variant of");

     "redhat linux"
association => a("is distobution of","redhat","has distribution");


Architecture::



    "$(linux_platforms)"
association => a("$(plat[f])","Linux","$(plat[b])");


    "$(solaris_2_platforms)"
association => a("$(plat[f])","Solaris 2.0","$(plat[b])");

    "$(solaris_10_platforms)"
association => a("$(plat[f])","Solaris 10","$(plat[b])");


Virtual_Machine::


"$(slogan_vms)"
association => a("$(virt[b])","slogan","$(virt[f])");



Application_Service::

    "jboss";
    "tomcat";
```

```
    "webservice"
association => A("$(sim[f])","@(webservices)","$(sim[b])");

    "apache_httpd"
association => A("$(prom[b])","@(apache_webservers)","$(prom[f])");

    "postfix"
association => A("$(prom[b])","vmDax","$(prom[f])");
    "DNS"
association => A("$(prom[b])","vmDax","$(prom[f])");

Package::
"openssl"
association => A("$(dep[b])","@(app_Service_Dep_openssl)","$(dep[f])");



Location::

    "room20_3_4_1"
association => a("$(loc[b])","slogan","$(loc[f])");
    "room20_3_4_2"
association => a("$(loc[b])","eternity","$(loc[f])");
    "room20_3_4_3"
association => a("$(loc[b])","dardania","$(loc[f])");



}

#############################################################################

bundle knowledge manual_entries

{
occurrences:



  Mark_Burgess::

     "mailto:mark@iu.hio.no"

            represents => { "Email Address" };

     "http://www.iu.hio.no/~mark"
```

```
            represents => { "Home Page" };

  slogan::

"cpuCount: 2, cpuSpeed: 2GHz, HardDisk: 300GB, hddCount 2,
Memory 3GB, networkCardt: 1, networkSpeed: 1Gbit/s, machine virtulized: True,
hasAnomaly: False"
        represents => { "machine resources" },
representation => "literal";

"InstallationDate: 20-03-2009, Language: English, patchLevel: 3_5,
LastTimeUpdated: 21-04-2009, virtualizationTech: xen,
virtualizationType: paravirtualization"
        represents => { "operating system" },
representation => "literal";


"Motherboard: x86, Processor: IntelCore 2 Duo, formfactor: Laptop,
vendorProduct: Dell Latitude E4300"
        represents => { "architecture" },
representation => "literal";

  vmDax::
"Memory: 500-1024MB, cpu: 2, cpuSeed: 1GHz, HardDisk: 80,
Licence: xxx-xxx-xxx, hasAnomaly: False"
        represents => { "virtual resources" },
representation => "literal";

        "http://os11.vlab.iu.hio.no/vmDax_monitoring.gif"
        represents  => { "Monitoring data" },
representation => "url";

  Robert::
"sittingPlace: R53, mob. 999-999, tlf.: 666-666, email: robertetmaildotcom"
        represents => { "contact info" },
representation => "literal";



}
```

```
#######################################################
# Bodies
#######################################################

body association a(f,name,b)

{
forward_relationship => "$(f)";
backward_relationship => "$(b)";
associates => { $(name) };
}

#######################################################

body association A(f,name_list,b)

{
forward_relationship => "$(f)";
backward_relationship => "$(b)";
associates => { @(name_list) };
}
```